INSTITUT D'INFORMATIQUE
FNDP NAMUR

THE TAILORING OF USER INTERFACES BY MEANS OF A COMMAND LANGUAGE

MEMOIRE PRESENTE PAR
GUY DAHMEN
EN VUE DE L'OBTENTION
DU DIPLOME DE
LICENCIE ET MAITRE
EN INFORMATIQUE

ANNEE ACADEMIQUE
1984/85

Wo bin ich ?

Wie bin ich bloß
hierher gekommen ?

Wie komme ich hier
wieder weg ?

<u>Die zentralen Fragen an</u>
<u>die Dialog-Schnittstelle</u>

(nach Nievergelt)

Where am I ?

How just did
I get here ?

How shall I get
out of here ?

<u>The central questions</u>
<u>to the user interface</u>

(according to Nievergelt)

Acknowledgments

First of all, I would like to thank Mr. Jean Ramaekers for having accepted to conduct this thesis.

I am especially grateful to Dr. Helmut Stiegler who gave me the first impulse to this thesis and whose advice was of a great importance in more than one occasion.

I am also grateful to Mr. Emmanuel de Cocquéau for his explanations on the SDF.

Last, but not least, I would like to thank all people involved from far and near in the elaboration of this thesis.

TABLE OF CONTENTS

Most of today's general-purpose systems are becoming more and more complex, and there seems to be no limit to this complexity. A user's access to a computer system and its various facilities is, in almost all cases, via a Command Language.

The growing complexity of the systems does not render the Command Language, and hence the user interface, less complex. The trouble comes from the fact that Command Languages, as well as their underlying functions usually are designed and implemented by computer professionals having each their own ideas about what a "user-friendly" user interface is, so that many different user interfaces are scattered throughout the whole system.

This situation is barely acceptable by these computer professionals. Now, due to the ever growing number of end users gaining access to the general-purpose systems, it becomes ineluctable to harmonise those user interfaces, i.e. to render them consistent.

Providing each user of such a system an own environment, consistent and tailored to his needs and requirements, would put him at ease and thus render the system more "user-friendly".

It is shown in this thesis that a command language can be used as a basis for the tailoring of user interfaces.

In the first chapter, the most important concepts used throughout this dissertation are defined.

The different ends pursued in user interface tailoring are gone through in chapter 2.

Chapters 3 to 8 treat the different means necessary to achieve user interface tailoring.

Chapter 3 discusses the language characteristics a Command Language should provide to support the tailoring of user interfaces.

Chapter 4 describes the guidance concept in the special context of Command Languages.

Chapter 5 handles command interface considerations.

Cometh next a chapter about the concept of a centralized, data-driven Dialogue Manager, which is a widening of one of the concepts defined in the first chapter, namely the Commmand Language Processor.

Chapter 7 describes the command editor, a function permitting to edit the data (objects) which drive the Dialogue Manager discussed in the chapter before.

Chapter 8 goes (briefly) over the concept of user profile.

Chapter 9 shows how the different means are to be used to achieve the ends enounced in chapter 2.

A few remarks before going into it.

It is assumed that the reader has some notions in the field of Operating Systems.

The systems we are primarily concerned with in this thesis are general-purpose time-sharing systems. This to justify why approaches like for instance the LISA one are not taken into account, because being infeasible for the systems discussed (although the LISA user interface is also a kind of Command Language).

As many other fields in the edp universe, human computer interaction is in the early stages of a science's development. As a result, the field contains principles which are sometimes contradictory and there is no consensus about the concepts used in it.

This is surely not surprising, as the main concern is on the user, i.e. a human being with all its pecularities and absurdities. How else could one explain the growing interest of psychologists in this aspect of human computer interaction?

Be that as it may, the concepts and principles used throughout this dissertation are surely defined and used otherwise elsewhere. If no satisfying definition was found, the one proposed is of course given while keeping in one's mind the goal pursued, namely the tailoring of user interfaces.

Even if the main basis for this dissertation was the way "it" was done in the Siemens BS2000's SDF, this is neither a user manual for the SDF nor an exact description of how "it works". The latter would anyway have been difficult to provide, as assuming that the reader knows the BS2000 is irrealistic and presenting the BS2000 is beyond the scope of this thesis.

Chapter 1: DEFINITIONS AND GENERALITIES

In this chapter, the most important concepts used throughout the thesis are defined: the user interface, the tailoring of user interfaces, the Command Language and the Command Language Processor. At the end come a few reflexions on the concept of "user-friendliness".

1.1 The user interface

"Interfaces keep things tidy, but don't accelerate growth: Functions do." (Alan J. Perlis, Epigrams on Programming)

This section begins with introducing the concept of user interface by showing the way it is defined in the literature. The second part gives the definition of the concept as used throughout the thesis.

1.1.1 Introduction

There are many (often informal) definitions of what the user interface to a computer system is.

According to Martin, the user interface is "the window through which the user sees the computer system". ([MART73])

For Parnas, the user interface are "the aspects or the system behavior that a user sees." ([PARN69])

Schofield et al. take another view: for them, "the 'user interface' consists of all messages that can pass between them [i.e. the user and the system] and the conditions under which they occur. To the system, the interface is fully defined, but the user can only rely on his expectations, developed during use of the system. He will describe an interface as 'friendly' or 'confusing'; to obtain his approval, the interface must be more than just a collection of ad hoc messages and conventions

- it must form a systematic whole." ([SCHO80])

Here is another view, taken from a 1980 Data Report: "die Benutzeroberfläche ist als Ebene zwischen Benutzer und Edv-system definiert - sie bestimmt den Grad der Nutzbarkeit und damit der Akzeptanz durch den Benutzer" ("the user interface is defined as the level between the user and the edp-system - it determines the degree of usability and therewith of acceptance by the user.") ([DATA80])

A definition close to the latter is given by Moran: "the user interface of a system consists of those aspects that the user comes in contact with - physically, perceptually, or conceptually. Those aspects of the system that are hidden from the user are often thought of as its implementation." ([MORA81]).

1.1.2 Definition

None of those definitions is obviously false but, except for the second one, they are far too general to be used in practice.

A better approach in my eyes is to start with the separation of the functionality of a computer system from its user interface (see fig. 1.1).

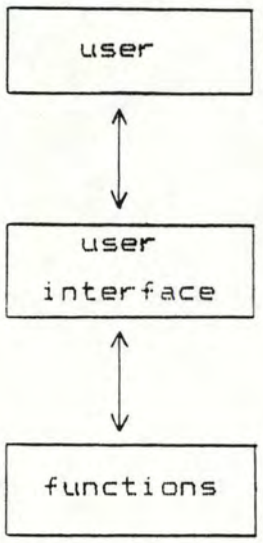


fig. 1.1: the user interface

The **functionality** of a computer system is defined by the set of functions (or tasks) the system is able to perform.

The **user interface** mediates between the user and the functions. From the user's point of view, it implements a **language** (i.e. a form of **communication**) that allows him to control the functions.

It is the way this language permits him to control the functions which determines the degree of usability and therewith of acceptance by him (e.g. a powerful application may loose or obscure much of its functionality if the user interface is not designed with care).

For this user interface to be "friendly", it must make the functions of the system **transparent** to the user.

The attentive reader will have noted that there is a bit of each of the definitions given above in the one proposed.

A language must be natural to use: natural for the user and natural to the function. When choosing the format for an interaction, the type of linguistic structures should be appropriate to the function and to the ability of the user. Here is where tailoring comes into effect.

1.2 Tailoring of user interfaces

This section begins by introducing the concept of tailoring by showing the way it is viewed in the literature. The second part defines the concept and discusses some generalities, while the third part presents the aspects comprised by user interface tailoring.

1.2.1 Introduction

Tailoring is seldomly discussed in the literature:

[BOTT78] and [BOTT82] discuss it in the context of command set tailoring for the IBM S/38, a workstation-oriented system.

[KUGL80] treats tools for the construction of "application-oriented interfaces, tailored to the semantic levels of the single users". This is an approach based on the concept of abstract machine.

[RAYN80] proposes to provide a tailored HELP environment for the different users of a general-purpose system.

1.2.2 Definition and generalities

Tailoring is defined as follows: the **tailoring** of user interfaces is the process allowing the definition of user interfaces according to the individual user requirements; this way, it should accomodate the user and render the system more "user-friendly".

Tailoring, due to the means to be used (see chapters 3 to 8), permits to avoid the non-uniformity of language features throughout the system; it also permits to hide inessential details from the user.

This is very important, as "... we are better off if ... inessential details are made INACCESSIBLE. So long as details are accessible there will be a temptation to use the knowledge of these details - for example to gain some local efficiency. In the long range context of program reliability and modifiability, however, such exploitation of detailed knowledge almost always has a net negative consequence." ([KUGL80])

As tailoring consists in fact in starting from a whole language and providing each user what he needs, it could be compared to the **view** concept used for data bases. However, tailoring goes further than providing views.

1.2.3 Aspects comprised

Tailoring comprises many aspects; I shall restrict myself to the aspects specifically related to the Command Language, which are the following:

- visibility of objects;
- different views of objects;

- individual command interfaces;
- forms of dialogue;
- response language;
- plurilinguistic aspects.

Other aspects are access rights, individual exception handlers,... ; I shall not treat them because on the one hand they have been discussed too many times elsewhere (as for instance the access rights) and on the other hand this would lead us too far.

1.3 Command Language

"A good system can't have a weak command language." (Alan J. Perlis, Epigrams on Programming)

The first part of this section introduces the concept of Command Language, this time by an excerpt out of the literature. It is defined in the second part, while the third part concerns the discussion of its relation to programming languages. The last part defines the concept of command procedures.

1.3.1 Introduction

The usual approach of people to Command Languages is best resumed in the following, extracted from [SCHN80]: "One of the most tedious tasks a programmer faces is using a control language to invoke operating system functions. Log-on procedures, password checking, file construction, compiler invocation, library usage, linkage editing, and device allocation require a special language which is **rarely designed for easy use.** Mention IBM's Job Control Language to a group of programmers and you will usually get a collegial smile indicating recognition of **shared anguish.** What makes these programmers so angry? Is the JCL bad, or is there something about it which produces unwarranted dissatisfaction? Can these languages be improved? Why have manufacturers persisted in using fixed or constrained formats with arbitrary and complex coding schemes?"

However, things have changed, primarily due to the fact that more and more non-computer specialists are confronted with their use.

1.3.2 Definition

For more and more users, a computer is not an end in itself, so it is to be viewed to them as a **tool**, a **servant**, and the command language is to be used to **command** this servant.

So, the term "Command Language" is to be understood in its broadest sense as providing the **outermost level of dialogue** between users and general-purpose systems. Thus they include both commands and responses.

Indeed, one is more and more talking about a common **OSCRL** (Operating System Command and Response Language); the reader interested in standardisation efforts should refer to [HILL83], [NEWM83] and [HOPP84].

The reader should bear this in mind while reading this dissertation: "gobbledygook"-JCL's are no more viable regarding today's user community, even if they are still used.

Whilst the Command Language is **primarily interactive** (in the past, manufacturer-provided JCL's emphasized batch use), it should be usable in a batch environment. However, I shall emphasize the interactive aspect and not treat the batch aspect in detail. This only to put things right.

A command is composed of an operation and of a set of operands (which may be empty), expressed syntactically as follows:

```
<command>::=<operation> ! <operation><operand>

<operation>::=<operation name>

<operand>::=<operand name><separator><operand value>
            [<operand-separator><operand>]

<operation name>::=<structured name>
```

```
<operand name>::=empty ! <structured name>

<separator>::=empty ! ' ' ! '=' ! ':'

<operand value>::=<actual value of operand>

<operand-separator>::= ' ' ! '/' ! ','
```

1.3.3 Command vs. programming language

One more important point arising from what has been said above is that a Command Language is **not** merely a programming language even if there are programming language constructs in it for the purpose of flow control.

There is one very important difference between using a programming language and using a command language. Using a programming language means to write my own ideas in some formal way, using a Command Language means I become incorporated into a very complex system I can never understand. A Command Language has to **hide** the complex system before me, which is much more than a programming language is doing and has to do.

Nevertheless, people have gone very far in the "cross-breeding" of command languages with programming languages, mostly influenced by the UNIX shell approach. Two striking examples are Ellis's LISP shell ([ELLI80], [LEVI80]) and the Command Language for the Ada environment ([BREN80], [KRAN82]).

Ellis's LISP shell is a command language embedded in LISP and running under the UNIX system. It is perhaps better described as an extended version of LISP designed to handle files, directories, etc., and to run programs written in languages other than LISP. In fact, such programs will in practice very commonly be system utilities as the editor, the directory lister, the off-line print, etc.

The MAPSE Command Language (MCL) blends features from the UNIX environment (such as IO redirection, pipes and background processing) with features of the Ada programming language (such as Ada-like parameter passing).

1.3.4 Command procedures

Commands can be grouped, to perform a given action, into command **procedures** which can generally be parametrized (similarly to a procedure call in a programming language). Command procedures are contained in files, sometimes with special commands to indicate beginning and end of the procedure (like in the BS2KDO), sometimes not (UNIX). They are generally called by issuing a special command.

The **name** of a procedure is always the name of the file containing the sequence of commands performing the given action.

Generally, commands contained in a procedure file must begin with the **command herald** (e.g. '/' for the BS2000); an exception to this rule is the UNIX system.

Examples of command procedures are:

- the shellfiles in UNIX: two possibilities are offered to execute command procedures: either to issue **sh** <procedure file name> (sh for shell) or to mark the file as "executable" and issue its name at command level;

- MIC files in TOPS-20 (Macro Interpreted Commands): these files must be of the type ".MIC" and called by **DO** <file name>;

- VAX/VMS command procedures: the file must be given the type ".COM", and the command procedure is called by issuing the file name;

- BS2KDO procedure files: procedures are enclosed between two commands, BEGIN-PROCEDURE and END-PROCEDURE, and they are called by CALL-PROCEDURE <file name>;

- there even exist micros with a (limited) procedure facility, as for instance the UCSD OS on the APPLE II.

1.4 The Command Language Processor

This section starts (once again) by showing the way the concept is shown in the literature and then proposes a definition. The third part treats the procedure and batch job cases of command processing, while the fourth part concerns a concept introduced in the part before, namely the command implementor.

1.4.1 Introduction

According to Beech, a language processor "is expected to understand correct utterances in the language, and to do its best in presence of errors." ([BEEC80])

For Jardine, "the command processor has the property of binding an application program to the Operating System." ([JARD75])

1.4.2 Definition

Taken together, both definitions provide a satisfying one; of course, the Command Language Processor has to analyse commands and "to do its best" in presence of errors. It should also provide the interface to the function implementing the command and call this function (called henceforth the **implementor** of the command) (see fig. 1.2).

According to this definition, during execution of the implementor, the CLP relinquishes control of the central processor, and it may gain it only if one of two things happen to the implementor execution: either the implementor terminates (normally or abnormally), in which case the CLP gains control at the level it had at the start of the implementor; or the implementor is interrupted (by some kind of a break signal), in which case the CLP gains only a restricted form of control. So, during the execution of an implementor, the CLP is **inactive.**

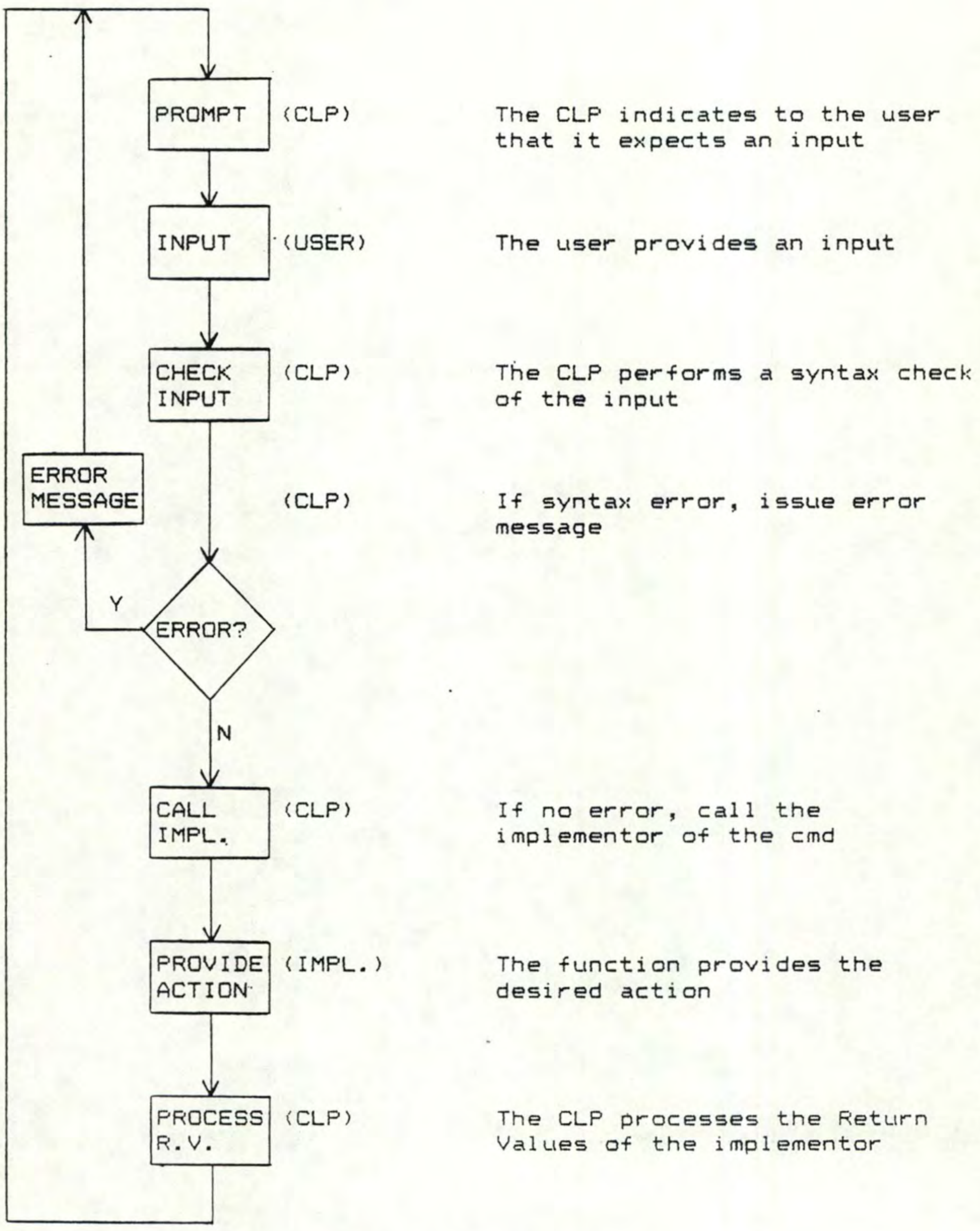

PROMPT
(CLP)
The CLP indicates to the user that it expects an input
INPUT
(USER)
The user provides an input
CHECK INPUT
(CLP)
The CLP performs a syntax check of the input
ERROR MESSAGE
(CLP)
If syntax error, issue error message
Y
ERROR?
N
CALL IMPL.
(CLP)
If no error, call the implementor of the cmd
PROVIDE ACTION
(IMPL.)
The function provides the desired action
PROCESS R.V.
(CLP)
The CLP processes the Return Values of the implementor

fig. 1.2: command processing (VERY broad view)

A question giving rise to controversy is whether the Command Language Processor is implemented as a **centralized** facility or is **diffused** throughout the system. According to [JARD75], "whether this is a conscious architectural decision, a design trade-off or an accident of implementation is a moot point."

OS/360 is an example of a diffuse command processor; it handles operator commands through a special interface (the Master Scheduler), Job Control Language through an interpreter (Reader-interpreter and Scheduler) and application program interfaces via subroutine calls or programmed interrupts (Supervisor Calls). ([JARD75])

Multics, UNIX, BS2000 and S/38 are examples of Operating Systems with centralized processors.

1.4.3 Procedure and batch job handling

In case of procedure or batch job handling, the CLP does not prompt the user for an input; input is read from the file containing the commands constituing the procedure or the batch job description.

In case of an error rendering impossible continuation of processing, the CLP either aborts the procedure or batch job or searches for a recovery point.

1.4.4 The command implementor

In the traditional approach, the implementor of a command is a system-level function; this means that, once the implementor has gained control, it is up to it to perform the dialogues with the user, be it in command form or other. One of the claims put forward in this dissertation is that application-level commands should be treated in the same way as system-level commands (at least in the way they are to be used by the user). Application programs are in fact functions other than OS functions.

In fact, application-level commands are commands correspon=ding to **sub-functions** of system-level functions.

For this reason (among others), the Command Language Processor concept will be **widened** in chapter 6, to the concept of a centralized **Dialogue Manager**.

1.5 "User-friendliness"

"The computer reminds one of Lon Chaney - it is the machine of a thousand faces" (Alan J. Perlis, Epigrams on Programming)

As the term "user-friendliness" is inseparably tied to the term user interface, I shall have to say a few words about it.

First of all, I do **not** believe that the term can be **precisely defined**; anyone has its own ideas about what "user-friendliness" is, and this is quite normal as anyone has in mind a different model of the system he is using.

Let us nevertheless look at the way it is shown in the literature; I think this will help to understand my position.

"User-friendliness" is a concept which. is often used but seldom defined in detail and there seems to be no consensus about its meaning.

Here is an excerpt from [DEHN81]: "Some authors regard user-friendliness more generally as an aspect of 'acceptance', 'user adequacy' or 'system quality'. Other authors just talk about 'usefulness', 'usability', 'user's satisfation', 'people compatibility', 'adaptation to human needs', 'ease of use', 'well-behaved system'".

Other authors go further and give a definition:

"User-friendliness is the ability of the system to react as expected by the user" (this one is not too bad);

"User-friendliness is the problem of facilitating the user's access to the computer" (this one is a bit too simple);

"In our opinion user-friendliness of a dp-system means, that all persons in the environment of the system are satisfied" (this one was given by philanthropists);

"A computer system is called user-friendly, if its equipment guarantees the psychical and physical well-being of the user, and if it provides job satisfaction and decreasing alienation" (same remark as above);

"For the user of a dialog system user-friendliness is defined system-theoretically as the linear coordination of the components input information and operation in the system" (wow!).

All of these definitions are also taken from [DEHN81]; there is a whole chapter for the reader interested in "user-friendliness" aspects.

The most "user-friendly" system would maybe be the one which **adapts** itself to the user. And yet! this would probably disturb certain (classes of) users. Anyway, we are still yards away from such a system, even if research is done (see for instance [GOOD84]: the dialogue is iteratively refined by hand based on the analysis of the user's behavior).

To conclude, the reader should bear in mind the fact that I do not believe that THE "user-friendly" system exists to date and that I do not claim the propositions made in this dissertation to be the most "user-friendly", even if they could render today's general purpose systems' user interfaces **more** "user-friendly".

Chapter 2: ENDS

In this chapter, the ends pursued in user interface tailoring are considered, beginning with discussing the existence of different types of users of a given system; it is shown that the definition of these types is not as straightforward as it could seem first. This fact influences the choice of our typing approach and the first two ends presented, namely the improvement of the initial training and the support of the evoluting user. Three further ends are then enounced, concerning plurilinguistic aspects, the introduction of a standardized system-wide user interface and the enhancement of security aspects.

2.1 Existence of different types of users

The first (and most obvious) reason for the necessity of user interface tailoring is the existence of different kinds of users of a system. Let us look first at the way these types are presented in the literature.

According to one approach (see [UNGE79], [BCS 78]), the users can (broadly) be divided into two major groups, the first consisting of those who use some general-purpose programming language, while the second group employs canned packages of several kinds. Whereas the first type of user interacts directly with the command language processor, the second one usually does not; this kind of classification corresponds to the approach commonly used, i.e. classifying users by **task**.

Another approach is taken by Ledgard et al. ([LEDG80], detailed in [LEDG81]), which classify users into 3 groups according to their **level of familiarity** with interactive computing:

Group 1: "inexperienced users" (less than 10 hours of terminal use)

Group 2: "familiar users" (between 11 and 100 hours)

Group 3: "experienced users" (more than 100 hours)
(Note that all users belong to one "task class"; in the case of the experiment presented, they were all students)

Further views are compared in [DEHN81] (pp.11-14), among which Martin, who distinguishes between trained and untrained, programmer and non-programmer, casual and dedicated users, with equivalence between (casual, untrained, non- programmer) and quasi-equivalence between (dedicated, trained, programmer). Another view presented is the view of Dolotta, who names three groups: end users, mid users and system users.

As is easily seen, there is no consensus in the literature about these user types; however, some important aspects should be clear:

- a casual user of one system can be a dedicated one of another system, and vice-versa;
- a programmer comes to use "non-programming" canned packages;
- even an expert user can "have a bad day", and this should not compel him to fall back upon user manuals.

Tailoring considerations should take all of this into account.

2.2 The typing approach chosen

Therefore, we shall base our typing approach on two concepts: the sophistication / transaction model proposed by Schneider et al. ([SCHN80]) and the role concept.

2.2.1 The sophistication/transaction model

The **sophistication model** relates the user's sophistication level with his experience of a language or system (see fig. 1.1).

The length of time a user remains at a given level is dependent upon:

1. the frequency of use of the language;
2. the language structure;
3. the language complexity;

4. the degree of experience with similar languages.

Remark: Progress is not guaranteed: a user can remain "forever" at a given level.

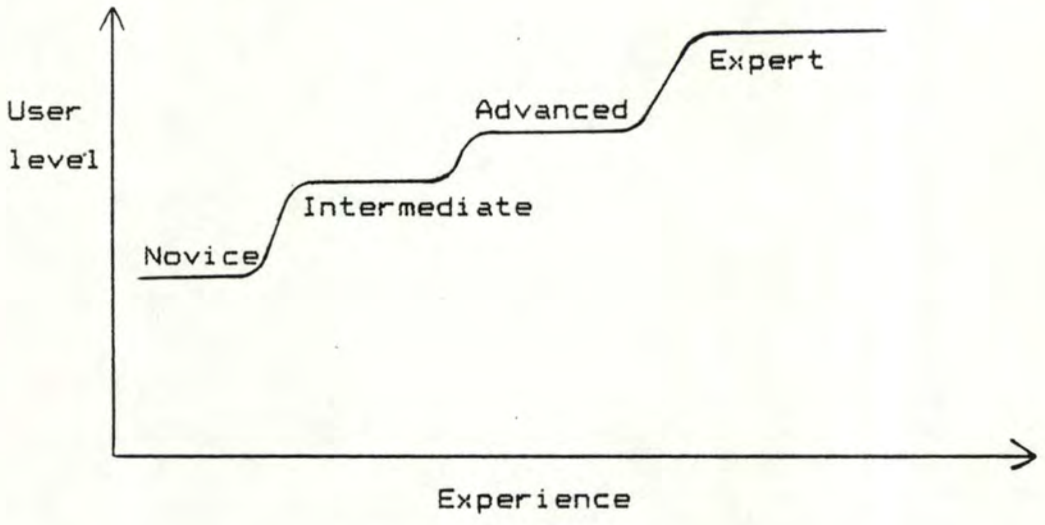


Fig. 1.1: User's ability vs. experience with a system

The model is independent of the user's task: he can be at the novice level in one feature, advanced in another. It is to be used together with what Schneider et al. call the **transaction model**, considering the stages required for the completion of a transaction:

Stage	Action
1	What function is to be used
2	How is the function used
3	How is the function coded
4	System responses
5	Evaluation of the response

Let us now look in somewhat more detail at the different levels:

The **novice** user is the beginner or the user having infrequent interactions with the system; for the beginner, guidance at stages 1 and 2 will be necessary, to first determine what function is required and then be guided in its use, whereas the infrequent user may know what function to use but may have forgotten its details.

After a period of time, the novice user will usually evolve into an **intermediate** user who may require little assistance in choosing a function or in its basic use. Help may be required in infrequently employed or complex structures, operands, or keywords. This user class will issue a command in as concise a form as possible.

The next step in the evolution is the transition from intermediate to **advanced** user (less drastic than the transition from novice to intermediate); the main difference between this two levels consists in the fact that the advanced user is concerned with groups of commands rather than single line constructions.

The **expert** user will employ all facilities of the system, and will be in a position to augment its functions through the design and development of new primitives and advanced programs.

The advanced and expert users are the ones which will add new commands to the system, the former "merely" by grouping commands and the latter by adding new functions.

2.2.2 The role concept

The second concept is the concept of **role** of the user, namely the function he executes in the system. Roles can be for instance <system administrator>, <application programmer>, end-user "based" roles,... and permit to define the responsibility of and actions performable by a given user.

The role notion in fact extends the model, as in a system there are persons which have very clearly defined roles, while others have not (in general, it is a matter of persons which lack enough knowledge of the system to be assigned a role). Similarly, for some persons, their role will change while evoluting, and for others not. However that may be, once a role has been assigned to a person, its interface can be precisely conceived.

In fact, the use of these concepts defines the **graining level** of the tailoring. While a system administrator will be somebody at the expert level, an applications programmer will

2.3.2 The "advanced novice":

In this case, there is only **one** important **premise** to be met (aside from the fact that his default mode of interaction should be the unguided mode), namely that the user interfaces to the different functions (for instance a new programming environment, a program for financial modelling, ...) present the same structure, similarly to the APPLE LISA approach. In this way, once knowing how to use one function, one can learn to use the others by comparison.

2.4 Support of evoluting user

This concerns the users which have crossed the novice level; as we have seen (see 2.2.1), these users require more help for executing a function than for choosing one. For them, the default mode of interaction is the unguided mode, and a **temporary guidance** mode is required. The latter is a mode in which guidance is provided just for the execution of one command, without having to change the guidance level explicitly (i.e. by means of a command).

One further argument for this temporary guidance is the following, claimed by users of interactive systems (see [NICK81], p.475): "Effective use of the system depends on knowing too much details." There is the point: effective use should not depend on knowing details, but on knowing how to get these details without loosing time.

2.5 Plurilinguistic aspects

This section, as own experience has shown, primarly concerns the rank beginner, because of acceptance problems from the side of the other users. Nevertheless, the latter are also concerned with this aspect, as they could get "help" information in their native language, while using the commands in the (traditional) English-based form.

Providing an interface completely based on the user's **native language** would relieve this user from having to learn English, still considered as THE basis for Command Languages.

2.6 A standardized user interface

As it is difficult to evaluate which is the primary user level of a given system (i.e. the level to which belong the most users), though it seems to be the intermediate level, according to [SCHN80], the interface handling should take all levels into account. This means that it should provide a given interface to all users (from novice to expert) as well as give the expert (and advanced) users -and those users whose role requires it- flexible, consistent means to build up their command interfaces (for their own use or for others).

Moreover, there is no reason why application-level commands should be handled in a way different from system-level commands (at least the way they are seen (and thus used) by the user).

All this is only made possible by providing a tailorable, generalized user-interface which can be imposed as a **unique, standard interface** throughout the whole system. This in fact is a typical example of a means becoming an end in itself.

A standard interface has several advantages:
for the user: uniformization of interaction throughout the whole system;
for the application designer: the time for designing a new application is reduced since he is relieved from outprogramming (and thus testing, ...) the interactions with the user; anyone who has ever conceived a user-oriented application knows how much time this takes)

2.7 Security aspects

A further end in user interface tailoring is the enhancement of security; the tailoring is one possible realization of the well-known **"need-to-know"** principle, which is described in a more detailed way in [SILB83], [DENN82] or [FERN81]. In short terms, this policy restricts information to those people who really need the information to do their job, and only the amount of information necessary for doing it.

This comes close to what has been said concerning the role concept: once a role is clearly defined, it is possible to provide the interface tailored to the needs of this role, and nothing more or less. To achieve this, it is necessary not only to provide only the subset of commands needed by the user. Even more, the commands themselves should be tailored to the needs of the user by rendering certain operands invisible.

Tailoring could also be used to resolve the problem of what could be called the **"sensitive"** commands, i.e. commands which have to be used with caution (e.g. DELETE-FILE, ...). It is clear that a novice user is much more concerned with this than another one and he should therefore be the only one who is "bored" with confirmation asking, ...

Chapter 3: LANGUAGE CHARACTERISTICS REQUIRED

In this chapter, the language characteristics required by the CL are discussed, beginning with its functionality; it is then stated that it should bear aspects of natural-language-likeness, that its operand names should be semantically meaningful, and most important of all, that it must be consistent. An outcome of the previous properties is the necessity of a powerful, "user-friendly" abbreviation facility, which is discussed next. At the end comes a section on responses.

The required language characteristics are also (the basis of) a set of guidelines for design and maintenance of a Command Language.

3.1 Functionality

The command names should have a relationship to their underlying function, i.e. commands should be given semantically meaningful names, describing (to the extent possible) what their underlying function **does.**

```
So, for instance (BS2KDO):
   COPY-FILE
   DISPLAY-FILE
   CONCATENATE-FILES
   SHOW-FILE-ATTRIBUTES
rather than (UNIX):
   cp - copy file
   cat - concatenate 2 files / display a file on screen
        (the trouble comes from the fact that the side effects
           of the cat command are used for the display case)
   ls - show file attributes
or, even better:
   grep - search file for pattern (this one deserves a prize)
```

People want the machine to **do** something for them, they are **action-oriented.** In short, as they want the system to perform some **action on** some **object**, the general form of a command should be the following: <action><object>, i.e. **<verb><noun>.**

Objects include passive objects (like files for instance) as well as processing activities (like programs, file-transfer,.). The difference between these objects should only appear to the user through the <action>-verbs chosen, and not on basis of a formal difference between commands (i.e. on basis of the seman= tics, not of the syntax).

A verb-object scheme is a dual-level hierarchy; Thomas and Carroll report ([THOM81]) that people rate hierarchically consistent command languages better than those that are not hierarchical. They found that people learn hierarchical command languages more quickly and that the frequency of some types of errors was reduced by using a hierarchical CL.

Similarly, Green [GREE79] showed that a "bigger" language with clearly exhibited structure was easier to learn than a "smaller" language with an inscrutable structure. This detracts from the usual <small is beautiful> approach taken for instance for the UNIX system.

A natural outcome of this requirement is that **overloading** is totally eliminated; a command performs one action on a single type of object. Overloading may seem to be a way of keeping the command set smaller, but there are subtle (and sometimes treacherous) distinctions to be made which are up to the user. This poses a burden on him which should be supported by the system. Moreover, a same command having different meanings in different contexts will make him feel at least uneasy.

Another outcome is that **"rattletrapping"** is also avoided. An example of commands used as rattletraps are the SET and SHOW commands in the VAX DCL.
For instance, the SET command ([VAX 81]):
Format:

```
    SET option
where the options are
    CARD-READER
```

```
[NO]CONTROL-Y
DEFAULT
MAGTAPE
MESSAGE
[...]
```

Purpose:
The SET command defines or changes, for the current terminal session or batch job, characteristics associated with files and devices owned by the process.

3.2 Natural-language-likeness

In [LEDG80], Ledgard et al. state that "an interactive system should be based on familiar, descriptive, everyday words and legitimate English phrases", while Green and Payne view English-likeness as one of the guiding principles in CL learnability ([GREE84]).

This means that the operand names should be chosen in such a way that they **reflect**, together with the command name, an **English-like phrase**, easier to learn (and retain) for novice users.

Needless to say, it is not possible to arrange all operands in such a fashion that the command reflects an NL phrase. Generally, it suffices to regard the first operands, which are the essential ones (concerning the object the command is acting against) and the most often used anyway.

For instance,

```
COPY-FILE FROM-FILE=file1,TO-FILE=file2,
```

which can also be used as follows:

```
COPY-FILE TO-FILE=file2,FROM-FILE=file1.
```

3.3 Semantically meaningful operand names

For all operands which can not be designed such as to reflect this NL aspect, the design rule is the following: the operand name is to be chosen in such a way that the **semantics of** the corresponding **value** can be **read off** from it.

For instance,

```
CREATE-FILE [...]
     ACCESS-METHOD=...
```

Of course, there should be a possibility of dropping these operand names or/and to abbreviate them. For the latter aspect, see section 3.5; the former brings us to the problem of whether operands can be specified by position, by keyword or if both should be allowed.

Only by position: this means that once a user begins to enter the operand values positionally, he must go ahead in the same way. This would be very error-prone: consider the case where he wants to give, say, the first and ninth operand of a command a value: CREATE-FILE hugo,,,,,,,,toto. Counting the separators is of course very amusing, but also very error-prone. So this way should be rejected.

Only by keywords: this will be boring once the user becomes accustomated to the use of the command. So, it should be rejected, too.

By position and keywords: this is the most "user-friendly" way to do it, but it should be handled with caution, and this for two reasons:

The first reason is that allowing the user to "switch" from positional to keyword-specification alternatively is error-prone, too, as it would compel him to know the order of the operands. So, once he begins to use keyword-specification, he should not be allowed to switch to positional specification:

```
COPY-FILE file1,file2,OPD-4=opd-4,OPD-8=opd-8,OPD-5=opd-5
```

The second reason is tied to the evolutionary aspect of any edp-system, and thus of the commands: operands can be added or removed. Problems can arise for instance with command procedures where operands have been specified exclusively by position. So, in command procedures, operands should always be specified by keyword; this seems straightforward in theory but is not in practice.

3.4 Consistency

This is in my eyes the **major property** any Command Language should possess, because it is a guiding principle for a novice user (he feels in control because consistency makes him comfortable; he is certain that unknown things will function like known ones) and because it keeps the Command Language from becoming unmanageably complex. The first part of this section discusses the scope of the required consistency and illustrates it by some examples, while the second part presents the means to achieve consistency. The next two parts treat in more detail the aspects of the solution proposed in the part before.

3.4.1 What kind of consistency?

Both syntax **and** semantics of the commands must be consistent.

First of all, the command names should be constructed in a consistent way, that is, always in the form <action> against <object>, i.e. <verb> <noun>. Similarly, the first operand should be the object the command is acting against (e.g. CREATE-FILE and DELETE-FILE should have as first operand FILE-NAME).

Second, command names should be **congruent**; congruence should exist on two levels:

- for one and the same object X, between actions performed on X: CREATE-X, DELETE-X;
- for two different objects X and Y, between equivalent actions: DELETE-X, DELETE-Y and **not**
  DELETE-X, ERASE-Y

Third, if you have the commands CREATE-FILE and DELETE-FILE, the operand name indicating the file to create / delete should be the same (e.g. FILE-NAME), and so on for all operands with corresponding meaning.

Fourth, defaults must be chosen in a consistent way, i.e. operands with similar function should default to the same value.

Last, but not least, if you have a command SHOW-X-ATTRIBUTES, with which the names of the specified object(s) is (are) also shown (this is yet consistent, as the name of an object is part of its attributes), the name of this object should be modifiable by using, say, MODIFY-X-ATTRIBUTES, but there should **not** exist a command like RENAME-X.

3.4.2 How can consistency be achieved?

First of all, **consistency rules** like the ones cited above must be **defined**; second, these rules must be **enforced** throughout the whole design and evolution phase of the entire system.

The ideal solution would be to have one **standard** Command Language (like the OSCRL) defining the consistency rules and to have in-house **committees** to enforce these rules during design and evolution of a given system.

3.4.3 Standardisation

Unfortunately, concerning the first aspect, the lecture of the different standardisation status reports gives rise to two impressions: the first is that they seem to be breeding up "little" monsters and the second that we shall have beards 'to the ground before something constructive will be realized and implemented.

Furthermore, **who** will be able to impose these standards? (the answer coming to one's mind is less than reassuring).

3.4.4 In-house committees

While waiting for a standard Command Language (?), in-house committees are necessary to maintain consistency; I know about three committees: DEC's DCL Clearinghouse, IBM's Usability Committee and Siemens KSK.

(1) One of the Command Language issues of DEC's DCL Clearinghouse is the DCL's consistency: "The overall consistency is verified by comparing the syntax with other

products having similar functions and / or a similar command syntax. The assumption is that a given function should be invoked with the same command and syntax wherever it is used in DCL (and vice versa). The basic philosophy is that the command set should be kept as small as reasonably possible." ([GRAY85]).

Alas! Keeping the command set small gave rise to commands like SET and SHOW; the SET command for instance is used to change defaults, characteristics of devices, "the current status or attributes of a file", ...; this is not very consistent.

(2) The philosophy behind <u>IBM's</u> <u>Usability</u> <u>Committee</u> (working on System/38) was that "usability must be designed into the system from the very beginning, and it must be as integral a part of the development process as performance, reliability, and serviceability. [...] [To achieve this,] the Usability Committee had responsibility for: [...] (3) Developing usability standards that ensure the consistency of both syntax and semantics throughout the various interfaces of the system including: the design of commands such that their syntax and semantics follow consistent rules [...]". ([DEME81])

In the System/38, each command is used to request a single operation on a specific type of object. So far, so well. Trouble starts with the construction of the command names (see [BOTT82]). They are constructed by concatenating abbreviations for verbs and nouns. Now, even if the different abbreviations are used consistently throughout the whole command set, the so-claimed "mnemonic" names seldomly have mnemonic power.

For instance:

CRT:	CReaTe
DKT:	DisKeTte
SBS:	SuBSystem
STR:	STaRt
RDR:	ReaDeR

There is little consistency in the choice of the abbreviations; you mean they always dropped vowels? No, as in:

USR:	USeR

OBJ: OBJect

DEV: DEVice

(although these ones are more mnemonic)

Note that apart from this, S/38's user interface really is consistent.

(3) Similarly, one of the purposes of the Siemens KSK (KommandoSchnittstellenKontrolle) is to design and maintain a consistent Command Language for the BS2000 (BS2KDO).

The **working method** of these committees seems to be fairly the same:

- proposition by working group;
- review/comment by committee;
- back to development group;
- discussion(s) between design group and committee until an agreement is reached.

The last point involves a fair amount of diplomacy and negotiation ([GRAY85] and own experience).

3.5 Powerful, "user-friendly" abbreviation facility"

The first part of this section states the necessity of an abbreviation facility (which is in the same time the reason why it is presented here, together with the language characteristics, although it is rather a characteristic of the Command Language Processor). The second part presents some techniques for constructing abbreviations, whilst the third one gives guidelines for the choice of the technique. The fourth part discusses the technique chosen for the SDF, and the last part treats some problems which could arise due to the use of the chosen technique.

3.5.1 Why an abbreviation facility?

As we have seen, the command and operand names as required by the characteristics (i.e. functionality,...) discussed above will be rather long, and compelling the user to always enter the full command and operand names will soon become frustrating.

Obviously, as stated by Benbasat and Wand in [BENB84A], "command abbreviations increase the general efficiency of the human user by reducing input time and entry errors. More importantly, the option of abbreviating increases the user **friendliness** of a system since, for frequent users, entering the full command name can become annoying."

3.5.2 Techniques for constructing abbreviations

There are a variety of techniques for constructing abbreviations, among which:

contraction: delete the vowels from a word; this gives rather strange-looking results;
abbreviations formed **by consensus:** the abbreviations are imposed, there is no real construction rule for the user. This compels him to know not only the command names but also their abbreviation;
mnemonics: see for instance section 3.4 (S/38); this is not very consistent. Furthermore the "mnemonic" power of these abbreviations is discussible (a lollipop for the one who tells me the meaning of DCLDTAARA, DSPSYSSTS or CRTDSPF);
choosing the names in such a way that they can be abbreviated to the **first letter**. This is the approach chosen by Ledgard et al. ([LEDG81]);
truncation: starting from the right end of the word and dropping off one or more contiguous letters until the desired abbreviation is obtained. Truncation can be "free" or with an imposed minimum (e.g. 4 letters for the VAX DCL).

3.5.3 Which one to choose?

Command abbreviation rules should be **consistent** and **simple**. Consistent meaning that the abbreviation rule can be stated unambiguously and simply, and simple meaning that it should be easy for the user to devise the abbreviation by a simple mechanical process.

According to this, the rule proposed by Ledgard et al. would be acceptable. Unfortunately, it is not very realistic in the context of a big system; Ledgard et al. made their investigations using an editor with a small set of commands.

Regarding the language characteristics discussed above, creation of command and operand names is not a trivial task. Imposing the constraint of first-letter abbreviation would make this task impossible. **Priority** is given to those characteristics.

3.5.4 The technique chosen: truncation

Therefore, the technique chosen for the SDF was "free" truncation (no minimum number of letters imposed), and this on two levels: on the level of each word composing the command (or operand) name and on the whole name.

So for instance,

    COPY-FILE

can be abbreviated to:

    CO-F   (because of **CR**EATE-FILE)

or to:

    COP

This rule is the more interesting the more words are used to construct the name (because this brings us back to Ledgard's first-letter abbreviation).

For instance:

    SHOW-FILE-ATTRIBUTES

gives

    S-F-A

Benbasat and Wand have shown in [BENB84] that the truncation method is the one the subjects prefered to use without trying any other forms of abbreviations (the subjects were only told they could abbreviate, but not how or how not to abbreviate). This means that **truncation** is the **most natural** way of **abbreviating** (the proportion of abbreviations using truncation was 80 percent).

3.5.5 Some problems

Problems can arise using the "free" truncation method, due to the existence of commands with similar beginning (e.g. CR-F and CO-F) and to the evolutionary aspect of systems (once again), when new commands are introduced which make previous abbreviations fallacious.

The solution to this problem is two-fold:

- in the case of ambiguous abbreviations (e.g. C-F), inform the user in the following way:
  AMBIGUITY POSSIBLE BETWEEN:
  <list of possible commands>
- provide a list of existing commands, on screen and in alphabetical order.

3.6 The case of the responses

The first part of this section tries to define what response are (in a rather informal manner), while the second one presents some guidelines for the design of "good" responses. Part three tries to resume the guidelines, and part four shows how to achieve the responses' characteristics.

3.6.1 What are responses?

Responses are in fact all **messages** sent to the user by the different functions he uses (or tries to use). According to Dean, "one reason that some computer programs or systems contain bad messages may be that 'message' has come to means a terse one-liner that people are not expected to understand without an explanation. In fact, there are guidelines for preparing documentation to explain messages. And manuals are written to reveal what messages often do not reveal - their meaning.

People want a computer to provide messages that explain themselves, that say what they mean. In fact, psychologically, the meaning is the message. A message whose meaning has to be explained does not communicate - it fails as a message.

How do we ensure that a computer's messages are useful to the people who receive them?" ([DEAN82]).

3.6.2 Guidelines for the design of "good messages"

Dean defines a set of **guidelines** for the design of "good" messages. I shall briefly enounce the ones which are the most important in my eyes. I believe them to be self-explanatory, but also sometimes rather "rules of thumb" rather than real guidelines. The interested reader should refer to ([DEAN82]).

"Do not make messages arbitrarily short"

"Identify the messages that people need"

"Anticipate people's expectations"

"Help people fit the pieces together"

"Do not force people to re-read"

"Put people at ease"

"Write messages well"

"Use vocabulary that is familiar"

"Use standard conversational language"

"Use standard punctuation"

3.6.3 "Summary" of the guidelines

What follows is not really a summary, rather my opinion about the most important features messages should provide.

First of all, messages must (of course) be **syntactically** and **semantically consistent**; this in my eyes summarizes many of the guidelines cited above.

Second, different **levels of verbosity** must be provided, with the most concise level providing meaningful messages (this is very difficult to achieve).

Third, all messages (comprising the different verbosity levels) must be **presented** to the user in a **consistent way**, throughout the whole system.

3.6.4 Achievement of the responses characteristics

To achieve **consistency**, the same solution as proposed for the Command Language is at hand, namely **in-house committees** defining and enforcing rules (standards).

To achieve the different **levels of verbosity** and the **consistent presentation**, the **handling** of the messages should be **centralized**. As will be seen in chapter 6, this is not straightforward at all.

Chapter 4: THE GUIDANCE CONCEPT IN THE CL CONTEXT

This chapter begins with the definition of two concepts used henceforth. The guidance concept is then defined, after what the requirements made to a guidance facility are enounced. The way of achieving these requirements is discussed afterwards. Section 5 presents an additional guiding aid, situated on a more logical level. The last section discusses some characteristics of the mask network composing the guidance facility presented.

4.1 Definitions

Before going on, I shall define two concepts used in what follows, namely the mask and HELP concepts.

4.1.1 Masks

The concept used here is the one corresponding to the standard layout form as proposed by the DIN ([DIN 84]).

A **mask** is a schema represented on screen, to be used for display and input of data.

A mask has 3 parts (see fig. 4.1):

the **head** with status information;

the **body** containing the information and input possibility (if any) corresponding to the task currently to be performed;

the **tail** with on the one hand a **control area** for the user (2 lines showing the possible inputs and one or several lines to enter the desired input); examples of control actions are commands to get the previous or next mask, cancel the current state, answer-ahead commands to bypass one or several masks,..; on the other hand an area for **responses** to the previous input.

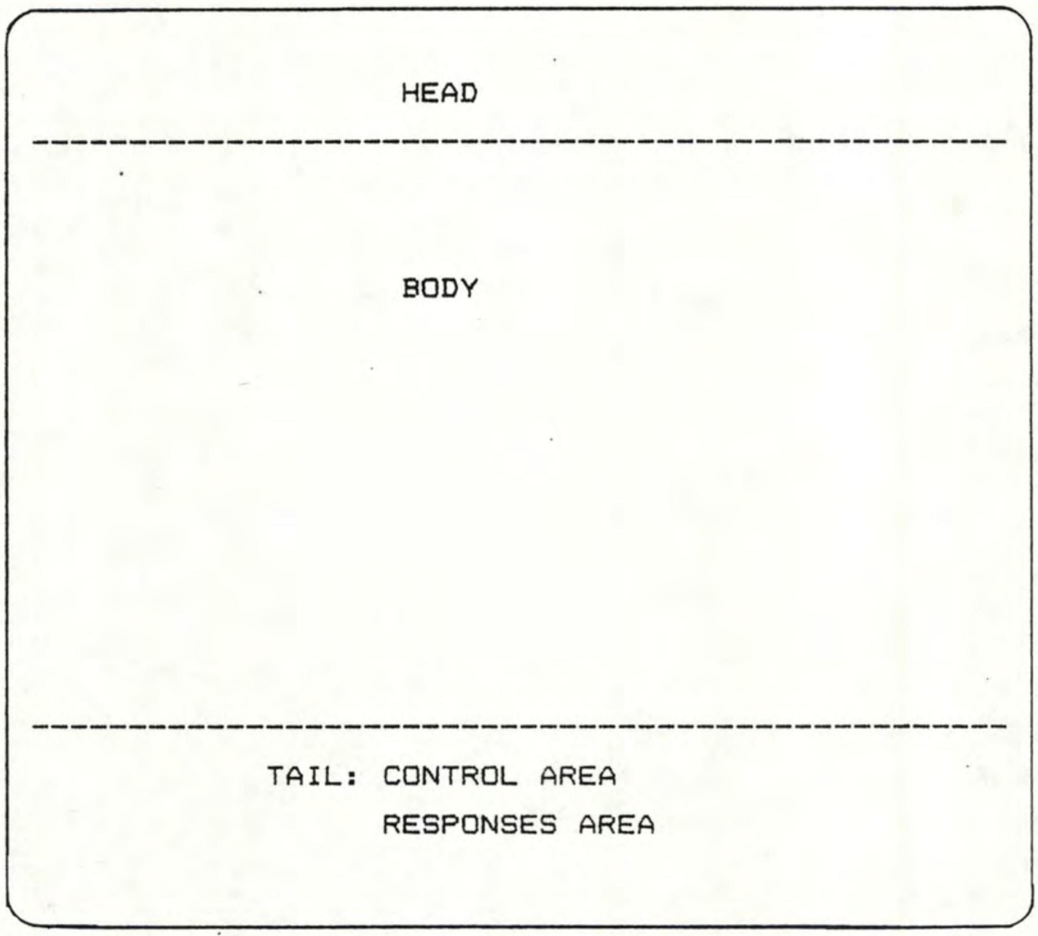


fig. 4.1: a mask

4.1.2 HELP

Help systems are now widely available on most mainframes, minicomputers and even a few micros. Online help systems provide a range of assistance from simple command assistance to elaborate and detailed tutoring ([HOUG84]).

The helpfulness of help facilities is often limited because help panels usually give general reference information rather than specific advice for the given situation, and they obliterate the screen that contains the input that is in error.

There even exist "HELP" systems needing a user manual on their own, as for instance TOPS-20's interactive user manual.

Really **helpful** help facilities should provide small increments of additional information that satisfy two **conditions:** <u>first</u>, they should be specific and relevant to the input that is considered and <u>second</u>, they should be displayable along with the input.

4.2 What is guidance?

The **guidance** concept it goes about here is what is generally called "computer-guided dialogue" (in contrast to "user-guided dialogue; see for instance [BENB84B]): the computer guides the user through the system, by proceeding stepwise from an initial situation (state) to a final situation (state).

The way this is usually done is by a question-answer session, where each couple question/answer is a step.

This works well in a system where the number of states is manageably small.

Unfortunately this is not the case in a big system; let us take an example:

```
[...]
WHAT DO YOU WANT TO DO?
create a file
UNDER WHAT NAME?
hugo
WITH WHAT PROTECTION?
?
```

So far, so well; but what about additional options (if any, are they desired or not?).

4.3 Requirements

Furthermore, the example implicitly assumes that the user knows what to do; as we have seen in the chapter ENDS, this is just what guidance is required for in the case of the novice user. (What function to use?). How to bring the user smoothly to use commands?

Moreover, the same guidance function should serve as teaching capability for the novice user and as temporary guidance capability for the others.

Briefly, the **requirements** are **two-fold:**
- guide the user in the **choice** of the command;
- guide the user in the **use** of the command.

4.4 How to achieve the requirements?

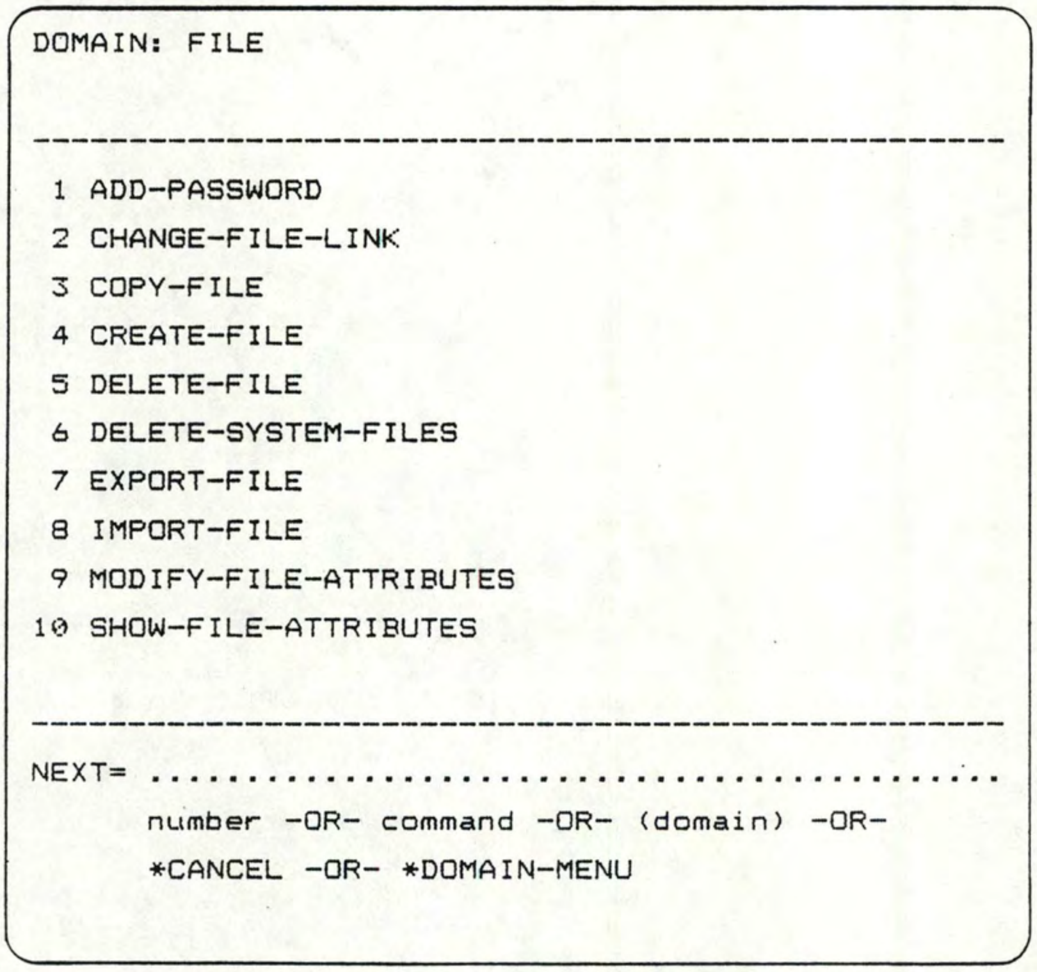
I shall present the way it was done for the SDF, because in my eyes it really achieves the requirements cited above. This section begins with discussing first the choice of the command, then the use of the command. The necessity of and presentation of different guidance levels is discussed in the third part, while the fourth part presents a particular kind of guidance, namely temporary guidance. Part five discusses on-line transitions between guidance levels.

4.4.1 Choice of the command

The user is guided in choosing the command by providing him the list of all available commands, in alphabetical order and sub form of a **menu** (presented as a mask).

A menu is a process whereby a set of numbered choices are displayed on the screen for selection by the user (see fig. 4.2).

For the menu approach to be sufficient for choosing a command, the command names must be functional. Additional information is provided under the form of an explanation of the function of the command.



```
DOMAIN: FILE

--------------------------------------------------------

 1 ADD-PASSWORD
 2 CHANGE-FILE-LINK
 3 COPY-FILE
 4 CREATE-FILE
 5 DELETE-FILE
 6 DELETE-SYSTEM-FILES
 7 EXPORT-FILE
 8 IMPORT-FILE
 9 MODIFY-FILE-ATTRIBUTES
10 SHOW-FILE-ATTRIBUTES

--------------------------------------------------------
NEXT= ..............................................
      number -OR- command -OR- (domain) -OR-
      *CANCEL -OR- *DOMAIN-MENU
```

fig. 4.2: a command menu

4.4.2 Use of the command

Once a command has been chosen, the user is provided guidance for the use of this command by **fill-in-the-blanks forms** (also presented as a mask): see fig. 4.4.

```
DOMAIN: FILE              COMMAND:SHOW-FILE-ATTRIBUTES


------------------------------------------------------
 FILE-NAME        = *ALL..........................
                    *ALL -OR- full-filename -OR-
                    partial-filename
 INFORMATION      = NAME-AND-SPACE..................
                    NAME-AND-SPACE -OR- SPACE-SUMMARY
                    -OR- ALL-ATTRIBUTES
 SELECT           = ALL.............................
                    ALL -OR- BY-ATTRIBUTES
 OUTPUT           = *SYSOUT.........................
                    *SYSOUT -OR- *SYSLST -OR- PRINTER


------------------------------------------------------
NEXT= ............................................
      *EXECUTE -OR- command -OR- (domain) -OR-
      *DOMAIN-MENU -OR- *CANCEL
```

fig. 4.3: an operand form

This clearly detracts from the usual HELP approach in that there is done more than just presenting information about the command and its operands: the same form shows how to use the command and permits to use it (by entering the desired values). Some operands may have sub-forms if one of their values introduces a structure (see section 5.2).

A very important aspect of user assistance is the use of **defaults;** it is a two-edged weapon: if the command language contains no default option, the user is forced to always enter maybe boring details. If too many default options are available, it can happen that a user can never find an error occurring in one of his command procedures (or batch jobs) because the system "fixes the bug" somehow - without the user knowing what is really going on.

Therefore, **defaults are always displayed.** This is crucial because it allows the user to see the default values and become accustomed to their being consistent. Displaying defaults allows a decision to be made prior to execution based on the defaults, not after execution.

4.4.3 Different guidance levels

Regarding what has been said in the chapter ENDS, different guidance modes and levels are obviously necessary.

There are two modes: the guided and the unguided mode.

The **unguided** mode provides two levels:

- the **expert** level: the prompt is the command herald, there is no support in case of errors; this level allows what has been called **parallel-sequential tradeoff** ([GAIN84]): the expert user may enter commands singularly or several in sequence, allowing him to speed up his interaction with the system as he becomes accustomed to the required sequence of commands;

- the "NO" level (which would better be called the "**advanced**" level): the prompt is "ENTER COMMAND" (and "ENTER STATEMENT" at application-level); there is support in case of errors by re-presenting the whole command up to the erroneus operand, together with an error message.

The latter level is interesting in case of command procedure processing, as it permits to correct an occurring error and thus avoids the procedure to be aborted automatically.

The **guided** mode provides three levels, differing in the amount of information provided to the user (**minimum**, **medium** and **maximum**):

For the operands:

- minimum: only the operands and the default values (if any) are shown

  e.g. FILE-NAME = *ALL.......

- medium: the different input alternatives are also shown

  ```
  e.g. FILE-NAME = *ALL.......
                   *ALL -OR- full-filename -OR-
                   partial-filename
  ```

- maximum: additionally, a help text is shown

  ```
  e.g. FILE-NAME = *ALL.......
                   *ALL -OR- full-filename -OR-
                   partial-filename
                   Name of the files about which
                   information is requested
  ```

For commands:

- minimum: only the names are shown
- medium,
- maximum: additionally, a text explaining the function of the command is shown

Moreover, the text specifying the allowed actions in the control area changes due to these levels.

Note that there is an <u>inconsistency</u> in the way these modes/ levels are modified: the **same operand** is used and to modify the mode and to modify the levels. In fact, from the users point of view, there are five levels (<u>but</u> still an unguided and a guided modes): EXPERT, NO, MINIMUM, MEDIUM, MAXIMUM. This will in my eyes disturb the user. The best approach would be to have the following command:

```
MODIFY-DIALOG-CHARACTERISTICS
        [...]
        INTERACTION-MODE=UNGUIDED(LEVEL=EXPERT,
                                        ADVANCED),
                         GUIDED(LEVEL=MINIMUM,
                                      MEDIUM,
                                      MAXIMUM)
```

(The underlined values are the default values)

4.4.4 Temporary guidance

As has been brought forward in the chapter concerning the ends (see section 2.4), a temporary guidance mode is necessary, i.e. a mode in which guidance is provided just for the execution of one command, without having to change the guidance level explicitly (i.e. by means of a command). Inputting a "?" at prompting level brings the user into the command menu (command choice); inputting the name of the command followed by a "?" brings him in the fill-in-the-blanks form corresponding to that command (command use).

4.4.5 On-line transitions between guidance levels

It is possible to interactively change the guidance level in two ways (the temporary guidance is a transition between modes of guidance):

by issuing a command changing the level explicitly for the rest of the user session (or until the same command is used again);

by inputting a "?" in the entry field of one (or more) operand(s), which will provide the information corresponding to the maximum guidance level.

Note that there is another inconsistency: the same feature (i.e. the "?" is used to get information of a higher guidance level (i.e. a change between **levels**) and to get into the temporary guidance mode (i.e. a change between **modes**).

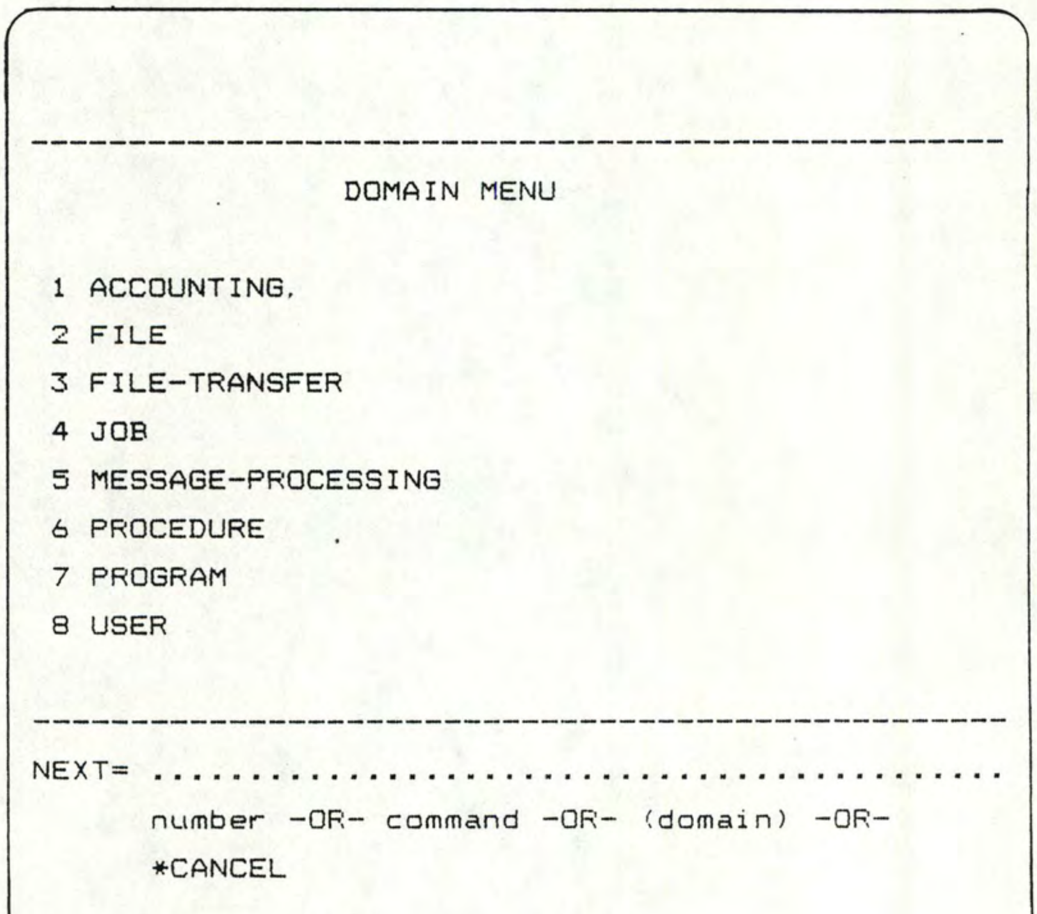
A possible solution would be to make the temporary guidance available when a **function key** is used.

Moreover, a feature that is lacking is the possibility of modifying the guidance mode or level anywhere in the system (to date, if one is in an application program, one must go to system-level to change the guidance level); this would help in hiding the system-level aspect to given users.

4.5 Command grouping

An additional guiding aid is provided by allowing to logically **group** commands into **domains** (e.g. FILE, JOB, USER,.); this is very interesting in case of a big command set. This grouping should be done in such a way that the commands contained in one domain define a kind of "**working set**" in which the user will stay "for a while". See for instance fig. 4.2.

If commands have been grouped, inputting a "?" at prompting level brings the user into the domain menu (see fig. 4.4), where he has the choice between getting into the command menu of a given domain or to the form corresponding to a command.

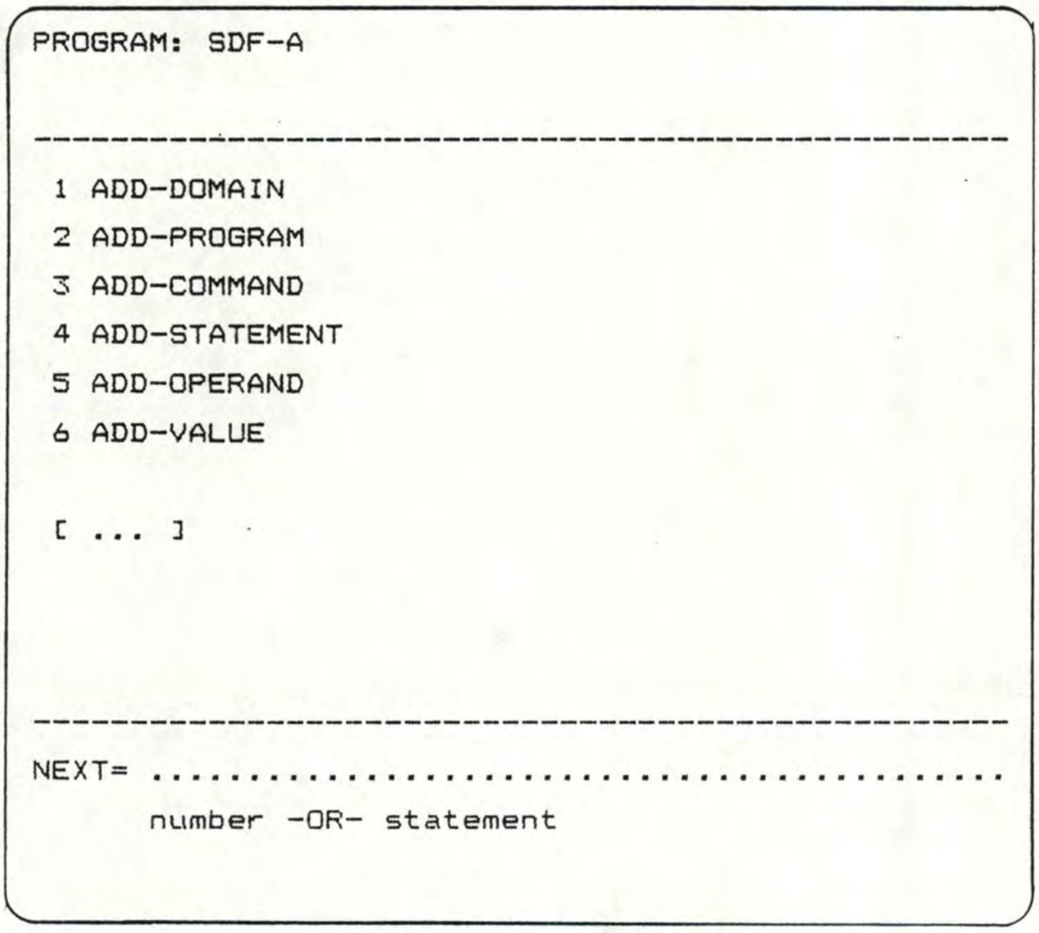


```
----------------------------------------------------------
                    DOMAIN MENU

 1 ACCOUNTING.
 2 FILE
 3 FILE-TRANSFER
 4 JOB
 5 MESSAGE-PROCESSING
 6 PROCEDURE
 7 PROGRAM
 8 USER

----------------------------------------------------------
NEXT= ..............................................
      number -OR- command -OR- (domain) -OR-
      *CANCEL
```

fig. 4.4: the domain menu

This grouping is only possible for system-level commands, as application-level commands are considered to be grouped under the name of the program they belong to.

This is yet another <u>inconsistency</u> as from the user's point of view, both groupings constitute working sets (and they are indeed presented in a similar manner to the user: see fig. 4.5), but the way of getting into and out of both is quite different (to get into a domain, one must enter (domain) in the mask's control area, while to get into an application, one must either enter START-PROGRAM <application-name> or APPLICATION if APPLICATION has been defined as command at system-level (by using the procedure concept).



```
PROGRAM: SDF-A

---------------------------------------------------

 1 ADD-DOMAIN
 2 ADD-PROGRAM
 3 ADD-COMMAND
 4 ADD-STATEMENT
 5 ADD-OPERAND
 6 ADD-VALUE

 [ ... ]

---------------------------------------------------
NEXT= .........................................
      number -OR- statement
```

fig. 4.5: an application-level command menu

4.6 Mask network

As has been seen in what precedes, the guidance facility presented constitutes in fact a **mask network** (see fig. 4.6).

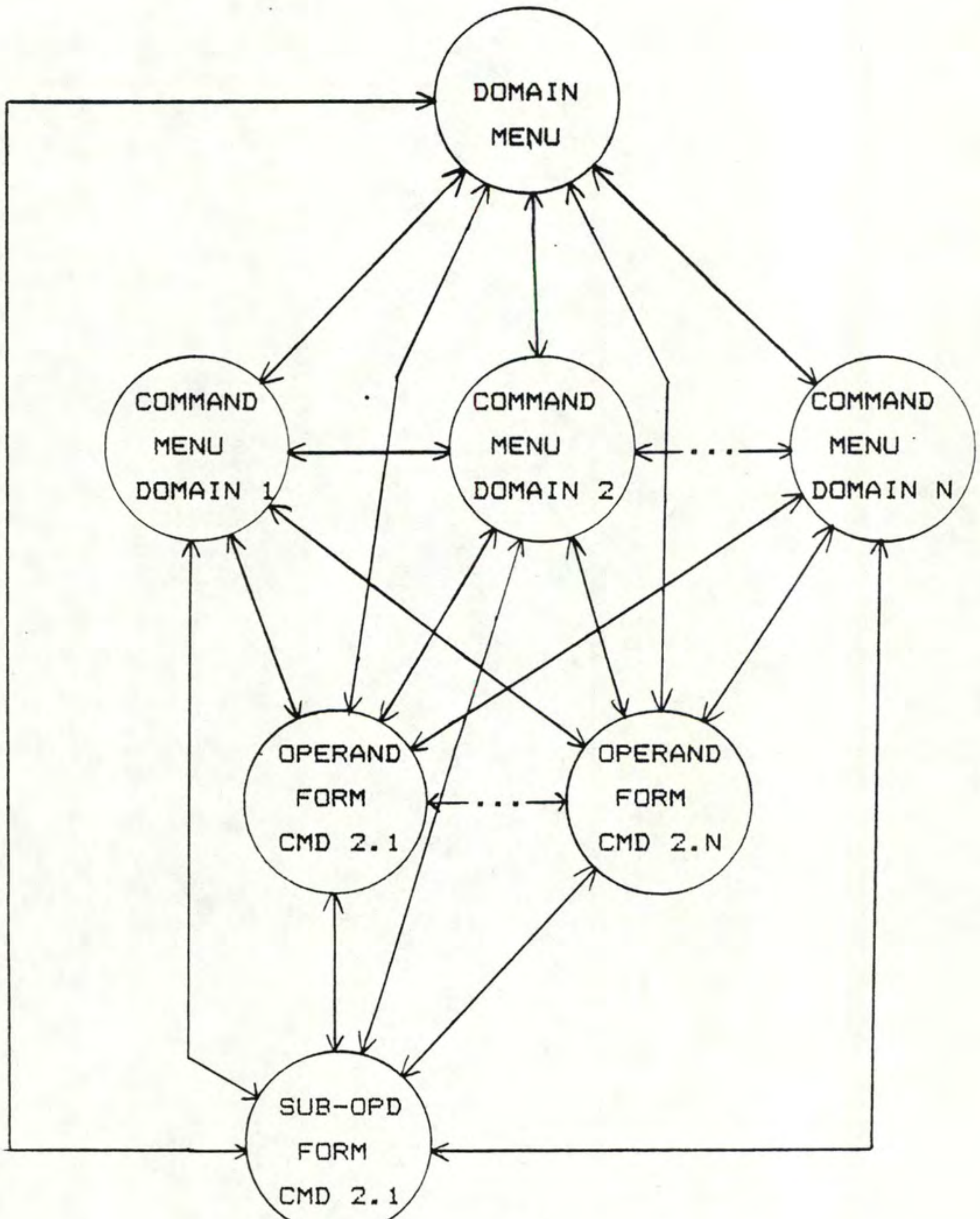


fig. 4.6 the mask network

Note: all of the states represented may be displayed on several screens, depending on the amount of information to be displayed. In this case, it is possible to get the previous/next mask by paging commands ('-' and '+').

The problem of "**getting lost**" in this mask network (mentioned in [BROW82]) is resolved by the status display in the head of the mask.

Moreover, at each state, it is possible to go back to a higher-level state by cancelling the current state (by using a command *CANCEL or by using a function key); at each state, it is possible to issue a command to be executed or not (i.e. followed by a "?") or a domain to get into, i.e. to issue an answer-ahead.

This answer-ahead feature makes that this guidance facility is **not merely menu-driven**, similarly to the ZOG approach, for instance ([ROBE81]).

Chapter 5: INTERFACE CONSIDERATIONS

This chapter deals with the separation of the external and internal interfaces of a command. It begins with general considerations about the interface specification of a command and goes then over to a closer look to the external and internal interface. Comes then a brief section about the representation of these interfaces, followed by the processing of commands as required by this separation of external/ internal interface. The most important steps are discussed in more detail in the following sections.

5.1 Interface specification

The interface specification in a command language mainly serves two purposes: it defines how a user may use the command and how actual parameters are passed through to the command implementor. The more information the interface specification contains, the more errors can be detected by the command language processor. As a matter of fact, one of the goals put in the foreground for the design of the brandnew command language for BS2000 was that the syntax-description had to contain as much as possible semantical dependencies (this will be explained in a more detailed way in the following section, concerning the external interface).

For flexible, powerfull tailoring to be possible, it is necessary to split the interface specification of a command into two separate parts, one describing the **internal** interface, as seen by the **command** itself, the other one describing the **external** interface, as seen by the **user**.

The definition given above is a short-hand definition (more detailed definitions are given below); it is interesting to note that the DIN NI AK 5.5. working on OSCRL standards gives this definition without detailing it and - what's more important - states that additionally, some informations has to be provided to map the external interface to the internal interface (though they propose to store it in the external interface).

The approach of splitting these interfaces allows the definition of different external interfaces to the same command for different (classes of) users (see fig. 5.1).

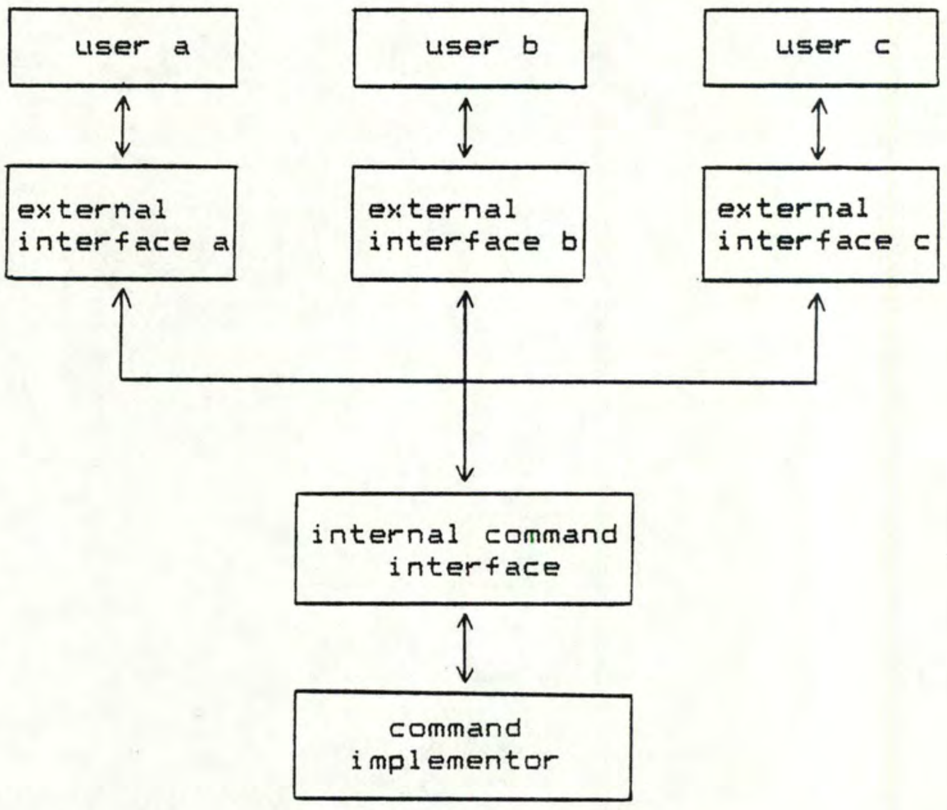


Fig. 5.1: External and internal command interfaces

Note that this concept clearly separates the concerns of the user and the concerns of the command implementor and that it is independent on how the command is implemented - as a program or a command procedure -; in any of these cases, the command language processor is able to control the specifications of the command, taking this task away from the executing module (as far as possible).

5.2 External command interfaces

This section begins by defining the concept and discussing some related generalities, and then presents the elements constituing an external interface.

5.2.1 Definition and generalities

The **external interface** of a command contains all informations causing effect on the user interface.

The external interfaces are to be used to restrict and tailor the use of commands for individual (classes of) users. Thus the concept of external interface allows user-dependent interface tailoring, and it provides a kind of syntax-oriented protection mechanism, allowing errors of the user to be detected prior to the execution of the command.

Using this approach, the introduction or modification of commands is possible without declaring command procedures (this is the "classical" approach, see for instance [SNOW84] for the HYDRA CL or [BOUR78] for the UNIX shell), thus avoiding to introduce high numbers of procedures for the only purpose of interface modification (e.g. for changing default values, for introducing default values in order to change a required operand to an optional one, for suppressing operands or restricting the range of allowed input values,...).

One final remark before talking about the constituents of the specification of an external interface: the only place in the literature where a definition of the external/internal interface was found is the already mentioned DIN... report; the trouble is that the specification given rests on too much concepts (resulting partially from the ad-hoc approach [sic] taken by the DIN...) than could be shortly described; therefore, I shall restrict myself to describe the constituents of the BS2KDO specification (the reader interested in the DIN... approach should refer to [DINN84]).

5.2.2 Constituents of an external interface

Here thus are the constituents of the specification of an external interface:

*)operation specific information:

- operation name;
- help texts (which may be language dependent in order to support users which are willing to use english commands but have a very limited vocabulary);
- mode of guidance (some commands are not allowed in the guided mode);
- accessibility (e.g. some commands may only be given in batch mode or out of command procedures);
- domains name(s) to which the command belongs (case of system-level command;
- program name to which the command belongs (case of application-level command);

Note: in the BS2KDO terminology, a distinction is made between a system- and application-level command, respectively "command" and "statement". As commands of both levels should be handled the same way at the user interface, there are mostly internal differences between them. I shall henceforth use the term **command** when what is said stands for both levels and **statement** when it is specific to the application-level;

*)operand specific information:

- operand name;
- default values;
- an indication if the operand is to be shown at the user interface or not;
- help texts;
- mode of guidance (e.g. very sophisticated features are not included in the guided mode for novice users but only on help for expert users);
- accessibility (see above);
- informations allowing the automatic generation of the masks for the menus used in the guided mode;

*)values specific information:

- type of the value with range indication if possible: an **enhanced typing possibility** has been introduced, merging the types required by a full screen manager with those required by an Operating System (e.g. integer, alphanumeric-name, filename, time, date,... for more details, see Appendix A). The value definition is not restricted to a single data type as "integer" or "keyword". Any fusion of - syntactically separable - data types may be allowed. This is more general than proposed in [FRAS83]. For instance, in order to make the semantics of alternatives clear very often keywords are used to indicate some kind of "meta-values" as well-known from fill-in-the-blanks forms, e.g. PARTNER-ADRESS= <address> or *AS-ABOVE. The most often found way of **type fusion** is combining keywords with other data types, since keywords are often used to indicate default values, e.g. *AS-ABOVE could be the default value for partner-address which is assumed by the system if no value is explicitly entered;

- an indication if the value introduces a structure (see below in the description of the operand tree);

- visibility of values in display and logging (e.g.passwords must never be logged nor displayed);

- a list of possible values (optional): if no value is given explicitly, the input is checked against the type (and range) of the value, otherwise the entered value must be within the specified list;

- an indication if the value can be overwritten dynamically by the implementor of the command;

*)the description of the operand tree (fig. 5.2):

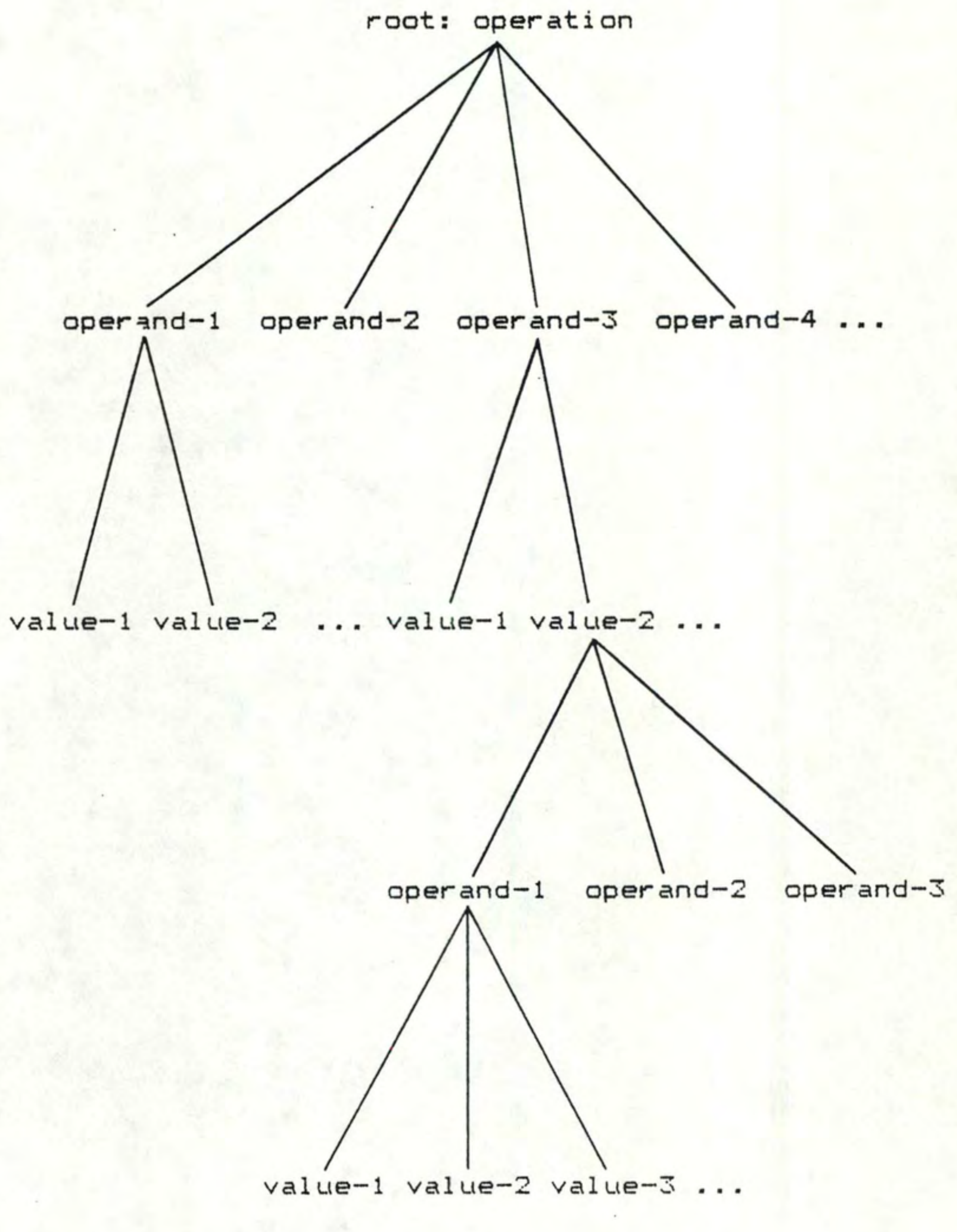

root: operation
operand-1 operand-2 operand-3 operand-4 ...
value-1 value-2 ... value-1 value-2 ...
operand-1 operand-2 operand-3
value-1 value-2 value-3 ...

fig. 5.2 : the operand tree

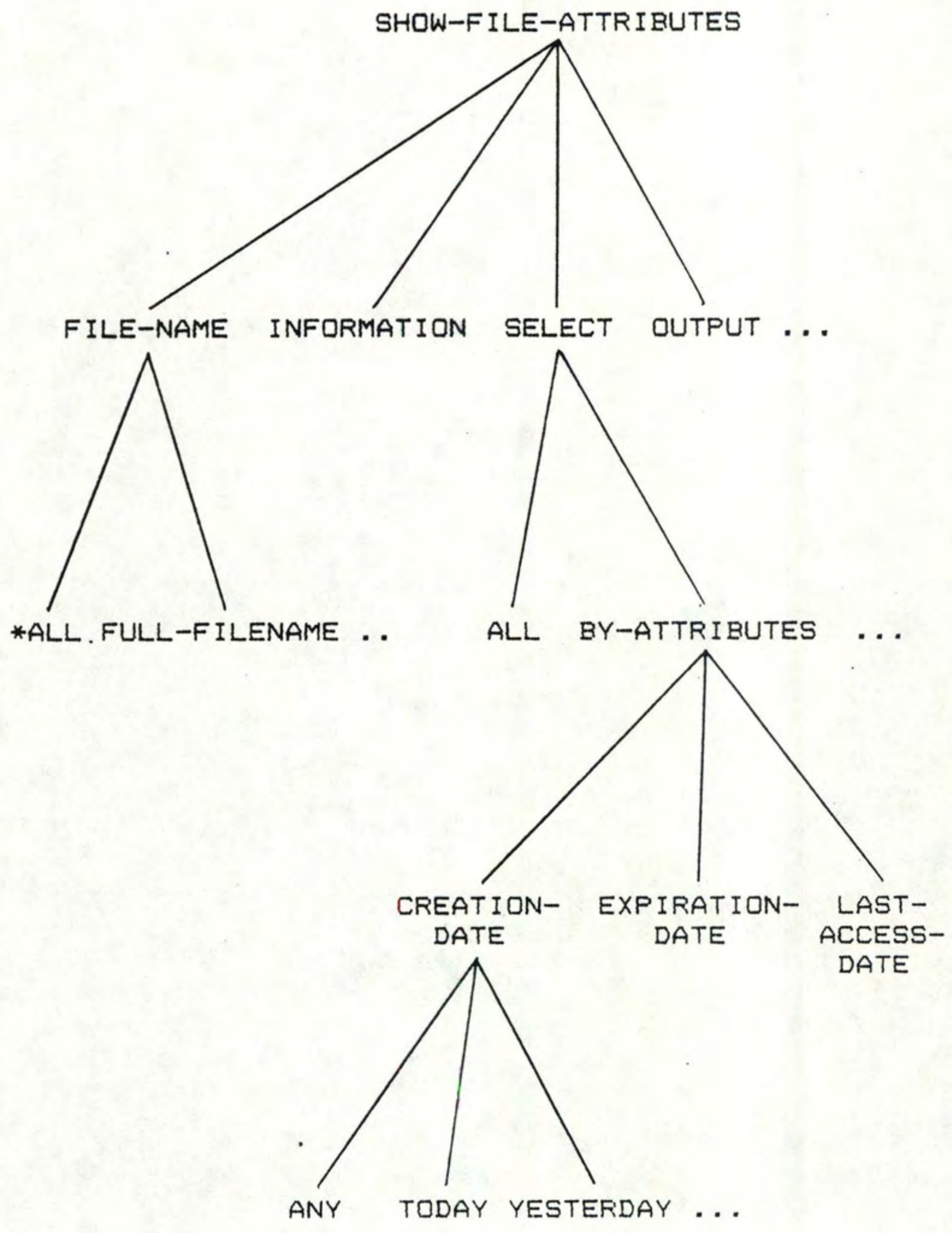


fig. 5.2 : the operand tree: example: command SHOW-FILE-ATTRIBUTES

- names (keywords);

- dependencies between operands: in classical command languages those dependencies can be checked either by programs [JOSL81] or by command procedures [BOTT82]; another approach has been taken here: the operands are arranged into a tree according to dependencies between them (i.e. all dependencies have to be reduced to tree-shaped dependencies). This is done by introducing a new syntax-element called "Struktur" (structure): a **structure** embodies several operands by putting them between brackets, expressing the logical dependencies of the structure-operands. What is more, structures can be hung immediately at a value (input alternative) of a given operand, making it strictly dependent from this value.
E.g. the commmand CREATE-FILE:

```
CREATE-FILE
   NAME          = <filename>

   [...]

   ,ACCESS-METHOD= SAM
                  ,ISAM(KEY-POSITION=...
                       ,KEY-LENGTH=...
                       ,SHARED-UPDATE=...)
```

In the example, the operands KEY-POSITION,... are only to be specified if the access-method for the file to be created is ISAM.
The operand tree of a command allows consistency between the values (input alternatives) of operands to be checked by the interface interpreter without requiring check-programs or command procedures;

- level of operands relative to the root (operation) or to the structure they are pending at;

- potential spanning of the tree by multiple values for single operands.

5.3 Internal command interfaces

The first part of this section defines the concept, while the second one presents its constituents.

5.3.1 Definition

The **internal interface** of a command contains all informations influencing the interface to the command implementor and the informations influencing the internal processing of the command.

5.3.2 Constituents of an internal interface

The specification of an internal command interface contains the following constituents:

*)informations concerning the **implementor** of the command, depending on the "type" of the implementor:

- if the command is implemented by a system program (i.e. Operating System functions, like for instance CREATE-FILE, COPY-FILE,...): specification of:
+) the entry,
+) the interface type (Assembler or other),
+) the type of the calling interface for reasons of compatibility (to old interfaces) and simplicity (there are cases where there is no need to express complex dependencies between operands, be it a brandnew created interface or not): OLD or NEW:
==> OLD corresponds to delivery in string format,
==> NEW corresponds to delivery in (new) structured format (see appendix)
+) the mode of logging (i.e. if the executing module or the command language processor itself has to perform the command logging),
[+) + several BS2000-specific informations]

- if the command is implemented inside of an application program (for instance utilities): no specific information required;

- if the command is implemented by a command procedure: name of the procedure; providing the possibility of calling command procedures by using an own command is interesting on two levels: first, one can, at the extreme, hide the procedure concept; second, the guidance feature is also available, which is above all interesting in the case where the procedure has many parameters;

*)internal names (which must be strictly identifying according to their level);

*)layout of the internal form as expected by the implementor;

*)access rights for modification; (e.g. certain commands, as LOGOFF, may not be removed);

*)residency of command descriptions (for performance reasons) : often used commands are kept resident.

5.4 Command descriptions

Even if there is a logical splitting between the external and the internal interfaces, physically, both are contained in one and the same **command description**, these command descriptions being contained in so-called **syntax-files.**

Command descriptions take up to 30K of space in the BS2KDO; this is an indication of the complexity and size of the underlying system more than of the command descriptions.

Syntax files can be created and modified by a special system function, which will be described in chapter 7.

Note : in lieu of talking about external / internal interface, it is also possible to view the command description as containing two kinds of information, namely syntax oriented information (corresponding to the external interface) and semantic oriented information (corresponding to the internal interface). Besides, this is the approach taken by H. Stiegler and the author in [STIE85].

5.5 Processing of commands

Before presenting the different steps, one important remark: the presentation given here is a broad view of the processing steps, describing the aspects specific to the separation of external / internal interface.

Here thus are the different steps:

(1) the name of the operation is resolved by (trying to) match it against a list of available (allowed) commands;

(2) the operands are checked for syntactical correctness; (possibly some protection checks may be done at this syntactical level);

(3) the default mechanisms specified in the external command interface are applied to expand the command entered by the user;

(4) the actual parameters are mapped to the formal

parameters of the internal form using the mapping mechanism as specified in the internal interface;

(5) the names of the operands are resolved;

(6) and (7) now and/or during the following command execution the protection mechanisms (if any) are applied (for instance access controls to files, ...).

5.6 Syntax check (step 2)

The basic principle for the syntax check is a **matching** operation between the syntax description of the command and the command entered by the user.

In the SDF, syntax check is done by using the operand tree, and this in a **breadth-first** manner.

Moreover, due to the enhanced typing possibility, the range checking and the expression of semantical dependencies between operands by means of the operand tree, some "**semantic**" checks can be done at this syntactical level.

5.7 Command expansion (step 3)

This is another kind of **matching** operation; if in the command entered by the user, an operand has not been explicitly specified by him, two cases are possible:

first, the operand is an optional one and thus has a default value assigned to itself; the command is expanded by this default value;

second, it is not optional, and the value must thus be asked to the user (the way this is done depends - in the SDF - on the guidance mode and level defined for the user: see chapter 4).

Note: the steps (2) and (3) could be done in parallel; whether this is the case or not depends on if one performs the mapping by starting from the command entered by the user (not in parallel) or from the command description (well in parallel).

5.8 Mapping (step 4)

This can be done either by "merely" concatenating the different values into a string or by performing a more sophisticated mapping. For the SDF case, see Appendix B for the description of the standard layout form.

5.9 Name resolution of the operands (step 5)

This means to check if the values of operands (which have not yet been checked at syntax level) correspond to **existing objects**, as for instance wildcard expansion for files.

This step is the one in the processing which is the most subject to controversy, namely **who** has to perform the name resolution: the Command Language Processor or the command implementor?

(a) the Command Language Processor: this is the approach taken by the DIN... and in a certain measure in the UNIX system, where the shell performs the wildcard expansion;

(b) the command implementor: this is the approach taken in the object-oriented IBM System/38 ([PINN78], [CONW78], [HARV78], [BOTT78]), and for the BS2KDO;

5.9.1 Pros and cons of the approaches

Before looking at the different systems, I shall give the pros and cons of the two approaches:

(a) if the name resolution is performed by the **Command Language Processor**, it will always be done in the same, **homogenous** way; moreover, it takes this task away from the implementor.

Unfortunately, this goes against any security aspects because implicating a strong centralization of perhaps sensitive informations.

An elegant solution to this problems is to have one specific module perform the name resolution for one given type of object (e.g. files) and to make this module available to all other modules needing it. This way, the homogeneity **and** security problems are solved, in a more flexible way than it would be by letting the Command Language Processor do the name resolution: as a matter of fact, if there is need to add a new type of object, letting the Command Language Processor do the task would compel to rewrite it in order to enable it to treat the case of the additional type of object, whereas the other solution only implicates to add a new module to the system, without having to touch at the Command Language Processor.

(b) if the name resolution is done by the **command implementor,** the security problems are reduced because each of the modules has only access to a part of the informations, possibly only in a given, reduced context.

There is one more argument for this approach, namely the case of the **distributed** system: here the name resolution has to be performed on the host, whereas the steps up to the mapping are to be performed on the terminal processor.

5.9.2 Comparison of different systems

Let us now look at the different systems:

First of all, the DIN... proposal: as it is not an existing system, it could seem curious to examine it; in fact, this has been done because it is the only place where a definition of the external / internal interfaces as well as a description of the processing steps related to it have been found. (see fig. 5.4)

As the proposal is aimed at big-sized systems, the homogeneity aspect is of real importance; yet the security aspect is of even greater importance. Anyway, the homogeneity problem can be solved by the solution we proposed, which in our eyes does not go against standard's requirements.

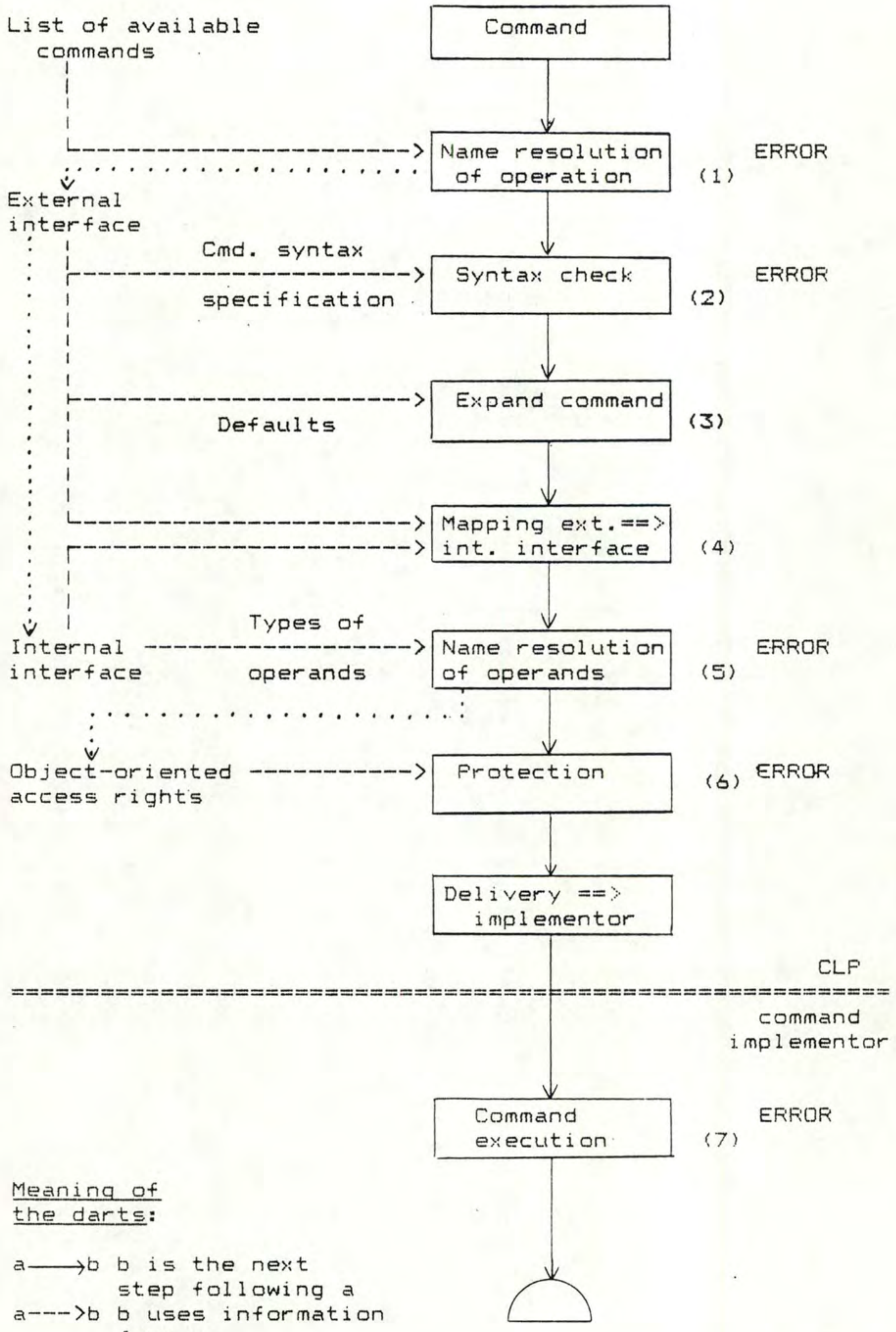
Next, the UNIX system: the first thing to note is that the shell does only perform name resolution in a certain measure;

this not so much because it "only" performs the wildcard handling (actually, directories are considered as files, I/O devices ara associated special files,...), but rather because it does the latter in a rather amazing way: the shell tries to expand the wildcards (if any) by attempting to match the files contained in the **current directory:** if any of the files match, the alphabetized list of matching names is delivered to the implementor; if not, the unchanged string is delivered to the implementor [CHRI83]. So, unlike the DIN... approach, no error dialogue is performed by the command interpreter for the case of name resolution.

Regarding this mode of name resolution, one could put the shell in a **third** category, namely where the name resolution is performed by the Command Language Processor and/or the command implementor (see fig. 5.5)

As the IBM System/38 is a workstation-oriented system, it seems quite normal that the approach chosen was to let the implementor perform the name resolution, even if the object-orientation could have allowed the Command Language Processor to do the job. (see fig. 5.6)

Let us now come to the BS2KDO: here the choice to let the implementor perform the task has been made above all because of the distributed-system aspect ([STIE84A]), together with the security aspect and the fact that it has been designed for an existing system (where the implementors already did the job). (see fig. 5.7)



<u>Meaning of the darts</u>:

a⟶b b is the next step following a
a--->b b uses information form a
a....›b a delivers info. to b

Fig. 5.4: Processing of commands, case of DIN...

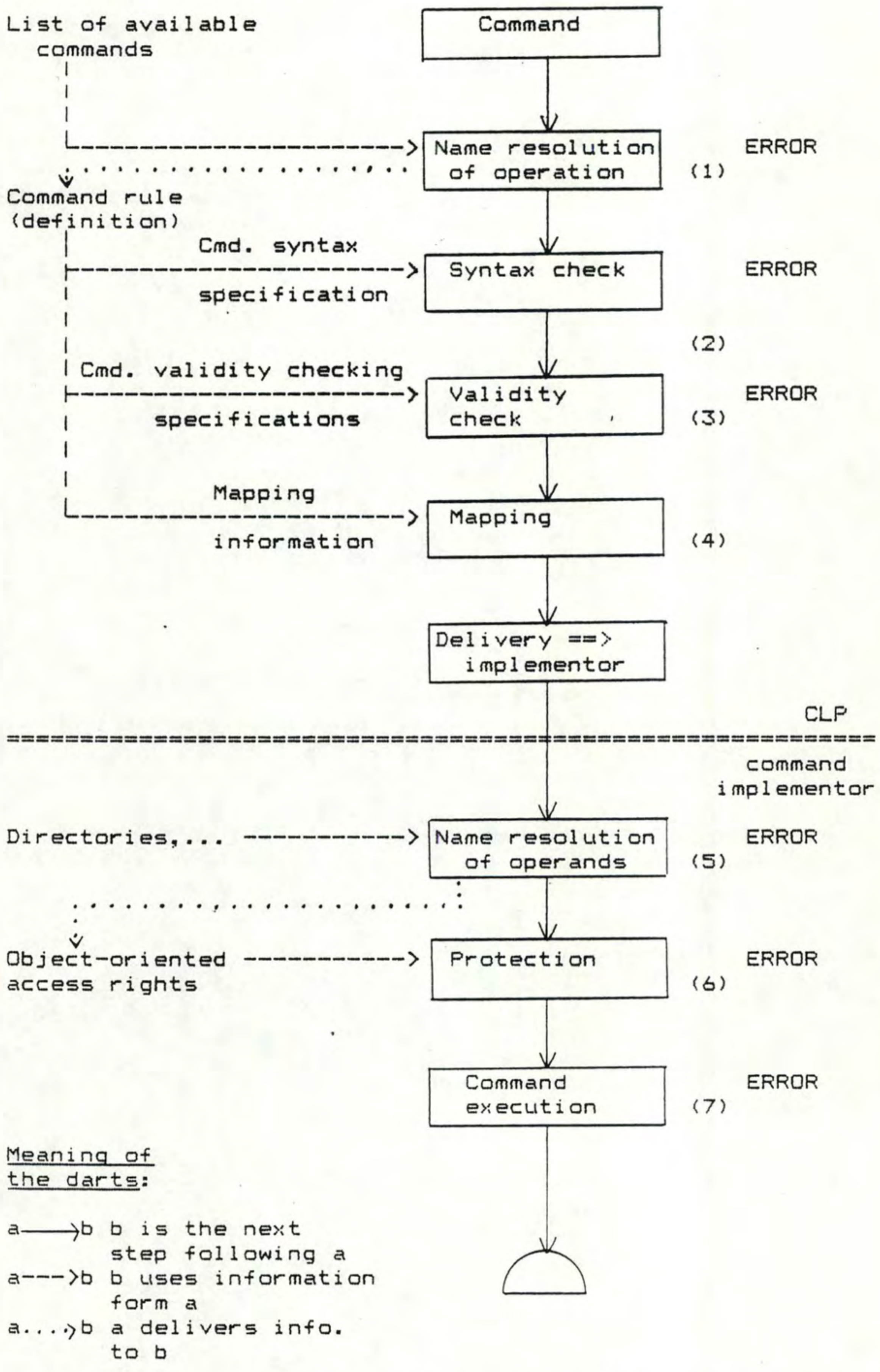
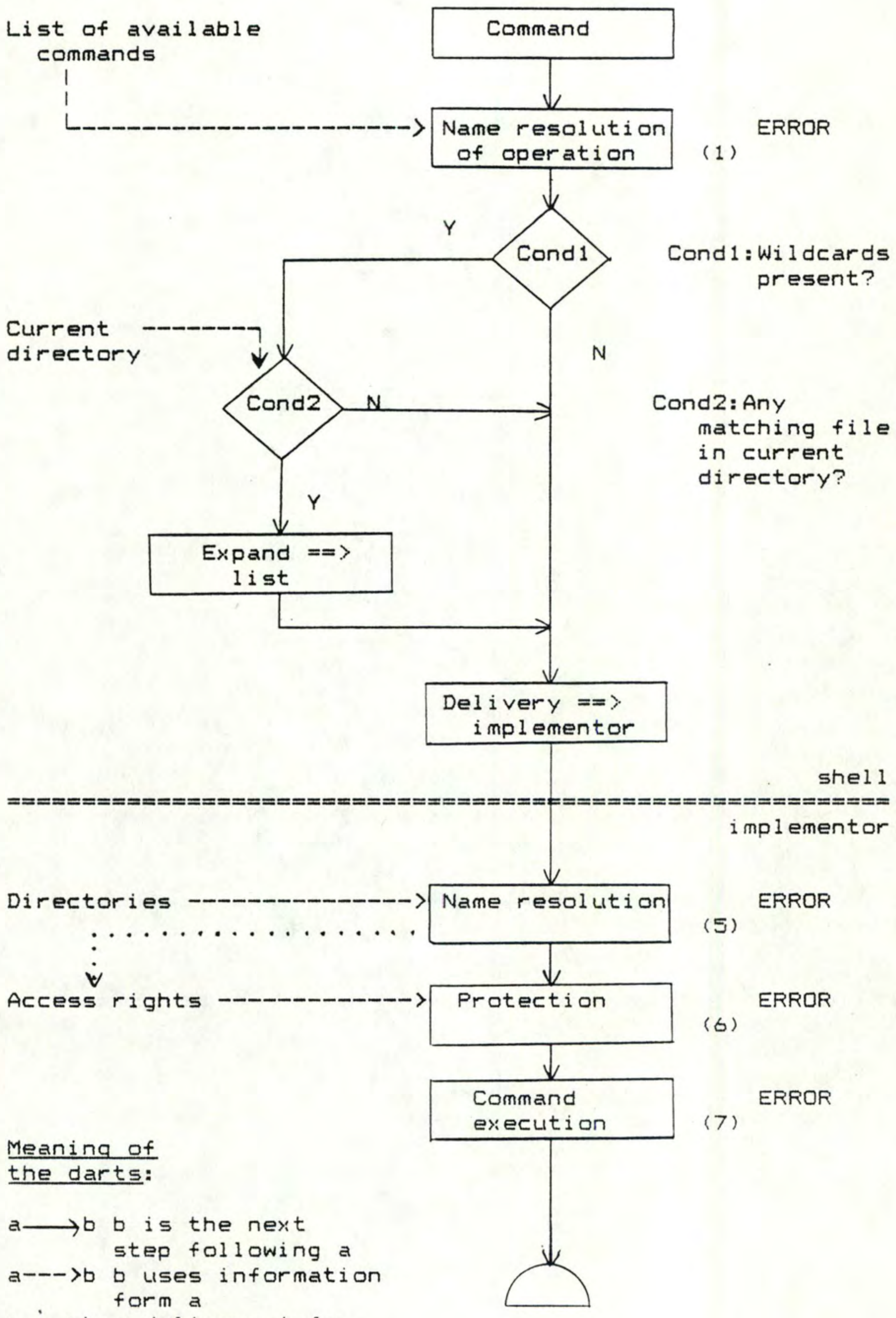

List of available commands
Command
Name resolution of operation
ERROR
(1)
Command rule (definition)
Cmd. syntax specification
Syntax check
ERROR
(2)
Cmd. validity checking specifications
Validity check
ERROR
(3)
Mapping information
Mapping
(4)
Delivery ==> implementor
CLP
command implementor
Directories,...
Name resolution of operands
ERROR
(5)
Object-oriented access rights
Protection
ERROR
(6)
Command execution
ERROR
(7)
Meaning of the darts:
a——>b b is the next step following a
a--->b b uses information form a
a....>b a delivers info. to b

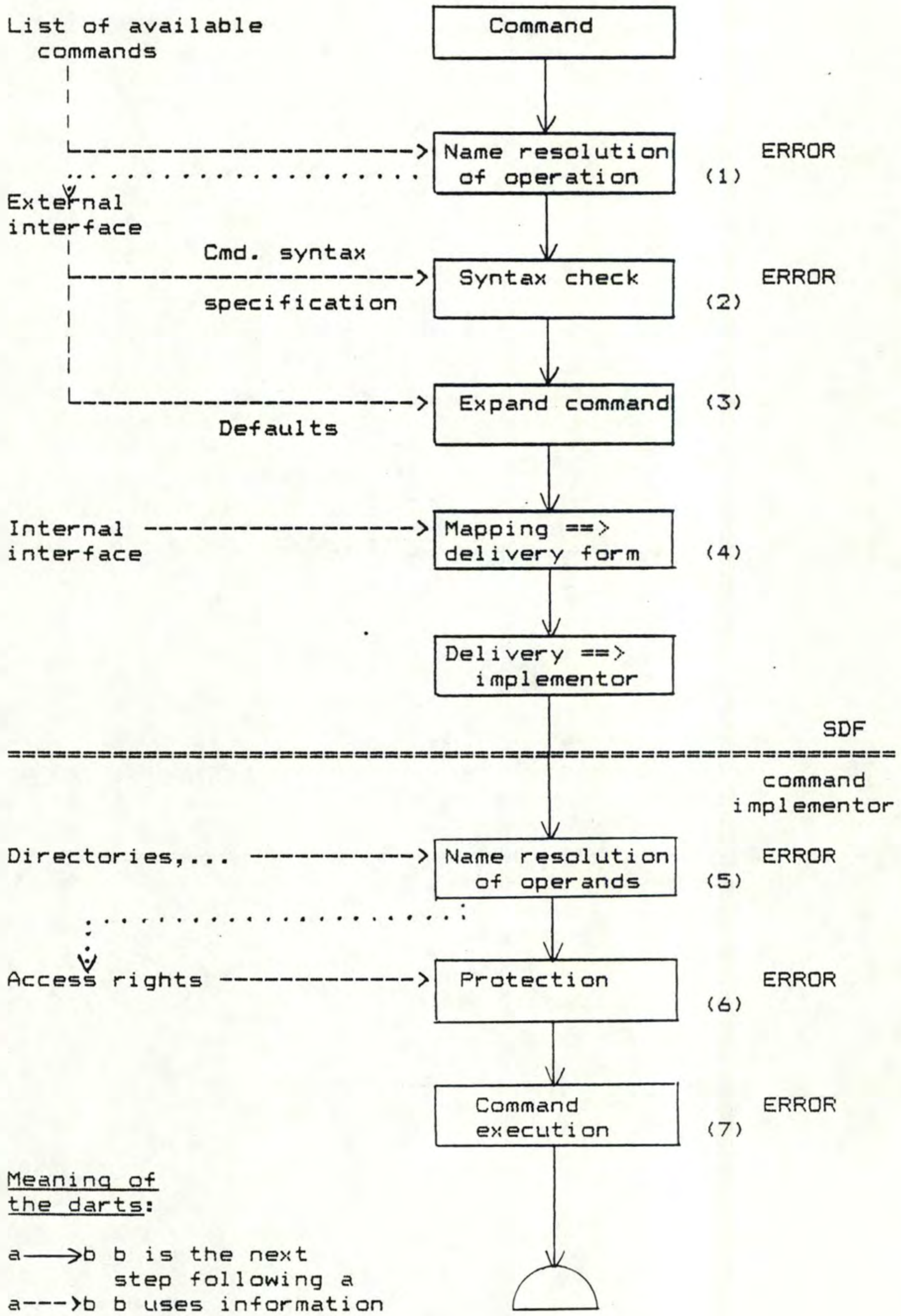
Fig. 5.5: Processing of commands, case of the S/38



<u>Meaning of the darts</u>:

a——>b b is the next step following a
a--->b b uses information form a
a....>b a delivers info. to b

Fig. 5.6: Processing of commands, case of the UNIX shell



<u>Meaning of the darts</u>:

a——>b b is the next step following a
a--->b b uses information form a
a···>b a delivers info. to b

Fig. 5.7: Processing of commands, case of the SDF

Chapter 6: CENTRALIZED, DATA-DRIVEN DIALOGUE MANAGER

The first section of this chapter briefly discusses the necessity of a Dialogue Manager, while the second one consists of a closer look at the features it should provide; their achievement is discussed next. Cometh then a discussion of the scope of control the Dialogue Manager should be allowed to perform, followed by some figures showing the size of the task of developing a Dialogue Manager and its associated Command Language.

6.1 Why a Dialogue Manager (DM)?

The capabilities required by the CLP to handle guidance clearly overstep the possibilities of what could be called a "normal" command interpreter, i.e.:

```
PROMPT
READ INPUT
SYNTAX-CHECK
IF ERROR
  THEN ISSUE ERROR-MESSAGE
  ELSE CALL EXECUTING MODULE
PROMPT
[...]
```

This is why the concept of CLP is widened, as yet announced in section 1.4.

Clearly, what is needed is what is now commonly called a **Dialogue Manager**, combining the features of a "normal" command interpreter with those of a full screen manager.

Actually, it should be seen as an **extension** to the Operating system, upon which any kind of function can call to perform the man-machine dialogues it currently requires.

To avoid misunderstandings, I shall henceforth use the term Dialogue Manager (or DM) in lieu of CLP.

6.2 Features of a DM

This section presents the different features a Dialogue Manager should provide.

6.2.1 Different dialogue levels

There is no agreement on the number of dialogue levels, but it is certain that a multi-level interface has a much better chance of pleasing its users than a single-level one. These levels should be consistent with each other; an example of a system with inconsistent levels is Mozeico's graphics system, using a five-level interface: one in a question-answer dialogue, another one in a tutorial frame-driven dialogue and the remaining three in a CL style ([MOZE82]).

6.2.2 Processing of sequences of commands

It should provide a means for processing **sequences of commands**, be it for a batch job, a command procedure or parallel-sequential tradeoff.

6.2.3 Availability to application programs

Its features should also be made available to **application programs**, such as to offer the same interface as is provided on system-level. Actually, as has yet been stated in the chapter discussing the ends, it should be possible to hide this "system-level" aspect from given users (depending on their role). So, similarly to the UNIX shell, the DM is not part of the OS, while the CLP is generally considered as being part of the OS.

6.2.4 Separation of user interface from function

Changes to the user interface should be handled in such a way that they do not require recompilation of the underlying program, i.e. the **description** of the **user-interface** should be totally **separated from the underlying function.**

6.2.5 Message handling

A means to handle the **messages** addressed to the user is also necessary. To discuss this point, I shall start from a typing of the messages needed by people interacting with a function, based on the one given by Dean in [DEAN82]. It is a typing **by purpose**, in contrast to a typing by audience, i.e. by the "receiver"; the audience aspect is (should be) provided by the different verbosity levels (see section 3.6).

Messages are of the following types:

report on the function's reaction to input (processing finished, progress display, results of processing,...)

report on the function's assumptions about input (e.g. assumed defaults)

request for a go-ahead

request to choose among alternatives (e.g. among actions to be taken, options governing processing,...)

request for missing information

request for correction of input

Moreover, it should be possible to **modify** the **verbosity level** and to "**switch**" these messages "**off**".

6.3 Achievement of the features

Let us now look at the way these features can be achieved; I shall once again present the way it was done in the SDF and compare it to other systems where appropriate.

6.3.1 Different dialogue levels

The different dialogue levels are provided by the different **guidance modes and levels** (see chapter 4).

The fill-in-the-blanks forms for the entry of the operand values in the guided mode serve a dual goal: first, to provide assistance for entering the operand values for a command; second, they can be used as **data entry panels**; (similarly to the screen forms provided by a full screen manager); this is made possible by the enhanced typing possibility (see appendix A and 5.2.2).

6.3.2 Processing of sequences of commands

The processing of sequences of commands is achieved by making the DM read its commands from a **logical file** which is assigned a given file (batch job or procedure file) or the terminal ("normal" command by command input or parallel-sequential tradeoff).

Of course, these groupings are only possible in unguided mode.

To enable parallel-sequential tradeoff, a **buffered I/O handler** is necessary: if the data required has already been input the DM reads this directly from an input buffer. If this buffer is empty the user is prompted for the input.

6.3.3 Availability to application programs

The DM facilities are made available to application programs (themselves yet called by a command) by providing two **programming interfaces** (macros): one to read and process a statement (processing steps 1->4, section 5.5) and another one to correct a statement.

While the purpose of the former is clear, the latter needs a few words of explanation: one of the design goals for the SDF was that **semantic errors** (resulting from the name resolution of operands: see section 5.5, step (5)) had to be corrected in the same way as syntactic errors (meaning they should be corrected interactively using the SDF). So, the correcting interface permits the implementor of a command to call upon the SDF to perform the error-handling dialogue(s).

The two interfaces provide the following informations for the DM:

The reading interface:

- internal name of the program to which the statement(s) belong;
- address of delivery area;
- a list of allowed statements (all statements of the program or a subset, sub form of a list of internal names);
- an indication for whether to ask for the statement name or to present the user immmediately the fill-in-the-blanks form of the operands of the statement; only used in guided mode, and in the case where the state= ment to be read is known (i.e. only one statement can be entered by the user);
- information for overwriting of defaults for operand values (if any are to be overwritten); defaults of several statements can be overwritten (those given in the list of allowed statements);
- specification of a message to be output by the DM (if any);

The correcting interface:

- address of delivery area (used as input where wrong operands have been marked) and as output (after cor= rection);
- overwriting of defaults: see above;
- message: see above.

Note: for the description of the delivery area, see appendix

B.

Two similar interfaces are provided for the system functions.

6.3.4 Separation of user interface from function

The usual approach to separate the user-interface description from the underlying function is to define the input syntax and the screen layout for each mask, thus having to specify sequences of masks, too. This specification is done either in a UISL (User Interface Specification Language), as in [ROBI85] or in a lower-level language, as for instance FORTRAN in [DIXO85].

This brings upon several problems: first, there is still an important programming burden on the application programmer. Second, there are problems with the help texts (comprising information on syntax of input and additional informations).

If they are specified separately, they must be very concise, as they constitute information to be displayed additionally to the one yet on the screen.

If, on the other hand, user-interfaces to the same function, differenciated by their verbosity level are to be provided, they must all be specified and thus stored separately.

In both cases, providing different, on-line modifiable guidance levels is difficult: in the former because too concise information is not enough to provide guidance, in the latter because "switches" from one mask specification to the other would cause consistency problems to the DM.

The approach taken for the SDF was to make it completely **data-driven**, i.e. **all** information required by the DM is provided by the data stored in the syntax-files:

- "general" information, as for instance the one to be displayed in the tail of the masks, the title of the domain menu,...;

- a list of available domains to generate on the one hand the domain menu and to perform on the other hand the name resolution for the domain names;

- a list of the available commands for the name resolution of the operation;

- information on the belonging of the commands to domains for the generation of the command menus for the given domains (if any);

- the command descriptions: for a closer look at the information provided by the command descriptions, see chapter 5. The different masks related to a command are generated **algorithmically** by the DM on basis of the command description.

Here another purpose of the **operand tree** becomes clear: it allows only mutually independent operands to display within one mask and to generate the correct order of masks according to the actual selection mode by a user. In other words, it allows the **automatic derivation** of **sequences of masks** (depending on the amount of information different numbers of forms have to be displayed).

This **one-data** description for commands covering all possible ways of user-interaction allows the user to switch into a different guidance level at any moment without causing any consistency problems for the DM.

This approach permits **prototyping** and **testing** of user interfaces even if the underlying (sub-) function is not yet implemented. Modifications to the user interface are easily performed by using the command editor described in chapter 7.

For a given user the currently valid syntax files are arranged in a **hierarchy** of three levels:

1. System-common (standard system interface),
2. User-group specific restrictions and privileges,
3. User private extensions.

Each higher one can overrule the appearance of a lower one, e.g. the name under which a command can be invoked. If a file of the third level is currently activated, it is searched for a given command first, and if the command is not found, the file of level 2 (if any) is searched through,...

Levels 2 and 3 permit **subsetting** and **supersetting** of the system-common command set (e.g. if several commands of the system-common set are to be made inaccessible to the user, they are marked "deleted" in the list of available commands of its user syntax file: subsetting; or commands are added in its user file: supersetting).

The main intent of this hierarchy is the saving of space; only the commands explicitly modified for a given user (group of users) are to be specified (and thus stored) entirely.

6.3.5 Message handling

As is easily seen, most of the message types are directly implemented by the guidance concept and the command descriptions (e.g. report on the function's assumption about input, request to choose among different alternatives, request for missing information,...).

A problem arises for the first type of messages, namely the report on the functions reaction on input.

The question is now: should the Dialogue Manager handle **all** messages to the user or not?

This is a very intricate problem, as on the one hand, the DM should handle **all** interactions with the user (and thus all messages), and on the other hand, these messages are to be defined at a semantical level (i.e. at the level of the function).

This problem seems **open-ended**, as letting the DM handle these messages imposes constraints on their syntax, form and size, and this is quite in opposition with the semantics aspect (which always involves a certain "ad-hoc-ness" and the messages to be jargon-free and tending to verbosity rather than conciseness).

The **modification** of the **verbosity level** and the "**switching off**" of the messages handled by the Dialogue Manager are provided by the different guidance modes and levels.

6.4 Why not let the DM do all of the job?

This would mean that the DM, in lieu of being called by the application to read and correct a command, once a given application has been called, stays in control, making it call the different sub-functions constituing the application.

This could seem more consistent when considering the similarity to the domain concept (see section 5.3): an application is also a kind of working set for the user.

Unfortunately, there are interactions between the different sub-functions constituing the application which can not be handled by the DM. So for instance, the use of a statement depending on the use of another statement or default values of a statement changing due to the use of another statement. An example of this is the command editor, described in chapter 7.

On OS level, the DM can stay in control because a **one-to-one mapping** between commands and functions is generally possible (at least should be), because it is a matter of very high level objects.

On application level, on the other hand, this one-to-one mapping is only possible **within** the application, and it is generally not possible to provide commands on system-level corresponding to each of this sub-functions. This because of the interaction mentioned above and because the objects are lower-level objects, often contained in system-level objects (e.g. command descriptions <-> syntax file).

6.5 Some figures

The figures given in this section will make clear that the task of providing a Dialogue Manager and its associated Command Language for a system of the size of the BS2000 is everything else than trivial. This may explain that the implementation of some concepts is not as neat as it should be. It is one more example for the fact that Software Engineering is very difficult to apply to real systems.

Planning, design and implementation of the Dialogue Manager took more or less 300 man/month, while the KSK spent 60 man/month to design the new Command Language, to which 120 man/month were added for the syntax file handling.

The Dialogue Manager counts 75 KLOC (Kilo Lines Of Code), and 1600 pages of documents were produced during the different phases.

There is an additional overhead for the whole CPU time of 2%.

Note that for the VAX, figures and overhead are similar.

(Stiegler, oral communication)

Chapter 7: THE COMMAND EDITOR

The command editor described in this chapter is the command editor of the BS2KDO. The first section of this chapter discusses the necessity of the command editor, while the second one defines the scope of edition (i.e. the possible actions). The next section describes the objects which can be edited, while the fourth one briefly addresses the problem of localizing objects. The last section describes the means provided by the IBM S/38 to act upon a command set.

7.1 Necessity of the editor

As has yet been stated earlier, tailoring considerations concern the aspect of offering a given, tailored environment to certain users (novice, intermediate) as well as allowing others (advanced, expert) to build up or extend themselves their environment.

What is more, these two aspects should (have to) be made possible by one and the **same utility**, this for (at least) three reasons:

first, to avoid the proliferation of command dialects throughout the whole system;

second, to assure syntactical and (to the extent possible) semantical consistency throughout the different hierarchies of command sets;

third, to enhance security considerations by making this utility the only tool allowed to handle syntax-files (by means of some checksum on the objects contained in the syntax-files).

In fact, the only difference between these two aspects should reside in the person who uses this utility: in the former, it is some kind of system administrator or "user-interface specialist" and in the latter, it is the user himself. Of course, the latter should have only restricted tailoring capabilities (regarding the reasons mentioned above), i.e. the use of the utility should itself be **tailored** to the needs and ability of its user.

7.2 Scope of edition

In fact, this utility is a **set of commands**, and the command edition constitutes a working set for the user (similarly to the domain concept presented in chapter 4).

Given this, the tailoring of the utility becomes easy using the utility itself, i.e. for a given user only a restricted set of editing commands is provided by using a "complete" editor at disposition of, say, the system administrator.

This working set comprises the following actions (the objects are described in the following section):

- show an objects characteristics;
- copy objects from another syntax-file;
- remove an object;
- add an object;
- modify an object's characteristics.

Of course, there are two further commands to open and close the syntax-files to be created or updated.

7.3 Objects which can be edited

All objects contained in a **syntax-file** can be edited. Regarding this, it could seem that the title of the chapter is erroneous, but the most important objects contained in a syntax-file remain the commands.

The objects are the following:

7.3.1 Global informations

These are of two kinds: those making part of the user profile (see next chapter) and language-dependent texts used by the DM (e.g. 'integer', 'filename', title of the domain menu,...). The former can be shown and modified while the latter can be shown, modified or removed for a given (natural) language.

7.3.2 Domains

Domains can be shown, added, removed, modified or copied from another syntax-file; their characteristics are their name, internal name and a (language-dependent) help text (optional).

7.3.3 Programs

Programs are in fact application programs; they can be shown, added, removed, modified or copied from another syntax-file; their characteristics are their name and internal name.

7.3.4 Commands

Commands can be acted upon in the following ways:

- show a command's characteristics: the following options are available:
  +) to show or not its operands and values;
  +) to determine the amount of information, which can be minimum, medium or maximum, corresponding more or less to the different guidance levels;
  +) to show the external and/or the internal interface;
  +) to determine which help texts to show
  (i.e. corresponding to what natural language).
  note: Appendix B shows an example of the information provided when a command's characteristics are shown by the editor.

- remove a command;

- add a command (external/internal interface);
- modify a command (external/internal interface).

7.3.5 <u>Operands</u>

Operands can be acted upon in the following ways:

- show an operand's characteristics: the following options are available:
  +) to determine the amount of information, which can be minimum, medium or maximum, corresponding more or less to the different guidance levels;
  +) to show the external and/or the internal interface;
  +) to determine which help texts to show
  (i.e. corresponding to what natural language).

- remove an operand;

- add an operand (external/internal interface);

- modify an operand (external/internal interface).

7.3.6 <u>Values</u>

Values are characteristics of operands, but also objects on their own, so they can be acted upon in the following ways:

- show a value's characteristics: the following options are available:
  +) to determine the amount of information, which can be minimum, medium or maximum, corresponding more or less to the different guidance levels;
  +) to show the external and/or the internal interface;
  +) to determine which help texts to show
  (i.e. corresponding to what natural language).

- remove a value;

- add a value (external/internal interface); because of the type fusion and the expression of semantical dependencies between operands using structures, several values may be defined for one operand;

- modify a value (external/internal interface).

7.4 Localization of objects

All objects except the global information must be **localized** to be <u>modified</u>; in fact, the localization is a positioning at the given object. For operands and values to be <u>added</u>, positioning in the operand tree is also necessary. This localization is also done using a command.

7.5 IBM System/38

In the IBM S/38, **three commands** are provided to allow a user to modify its command set: one to create a commmand (CRTCMD), one to delete a command (DLTCMD) and a third one to modify the attributes of a command (CHGCMD). ([S/38??], pp. 385-394)

Chapter 8: USER PROFILE

There must be a means for **matching** a tailored user-interface to a given user; this is made possible by the concept of **user profile**. It is a logical concept, i.e. not to be seen as a physical entity. It specifies as well the actions a given user is to be allowed to perform as the way he is to use the system and the way the system supervises him. These three aspects are discussed in the first three sections; the fourth section concerns the modification of the user profile. The last section shows how the concept is constituted in the IBM S/38.

8.1 Allowed actions

They are defined by several things:

first, the command/operand/value set as provided by the syntax files assigned to the user;

second, the name of this syntax file, as it has to be activated for this user at logon-time;

third, at the extreme, if the role of the user confines him to a given application program, the name of this program should be indicated to be started at the end of the logon-processing.

Note the duality at the level of information provided by the syntax file: as well its name as its contents.

The definition of the actions allowed for a given user defines the **initial context** for this user, i.e. what objects he can act against and what actions he can perform on these objects. This is made easy to realize thanks to the functionality aspect of the command names. In the case of a heavily overloaded language it would be more difficult, if not impossible.

8.2 Way to use the system

This includes the following aspects:

- the default guidance mode and level;
- the default temporary guidance level if the default guidance mode is the unguided mode (to date, the default temporary guidance level is always the MAXIMUM guidance);
- the default values for certain operands;
- the language the user wants for the help texts associated to commands and operands;
- the natural language to be used for the command/operand/ value set.

8.3 Supervision

The way the system supervises the user is determined by the way of **logging** the commands: if it has to be done at all, if the command/operand names are to be expanded, if the logging is only to be performed in case of erroneous commands or not,...

Logging can be used for an error analysis of the command use, which in turn can be used to test the Command Language's ability to meet the user's requirements.

According to Davis, "Command language-based systems are amenable to a detailed error analysis in a way that programming languages can not be. If such an analysis is coupled with a formalized task analysis (also better suited to Command language systems), one can be used to predict the other generating many testable recommendations." ([DAVI83])

8.4 Modification of the user profile

Some elements of the user profile can be modified **temporarily** (i.e. only for the current user session), as the guidance mode or level, the language for the help texts. The user can even activate another syntax file (if his command set comprises the command to do so), so he can for instance switch from an English-based command set to a German- or French-based command set.

Other elements can be modified in a **lasting way**, so the default values of the operands, the elements that can be changed temporarily (see above). These modifications will come into effect at the beginning of the next session.

The question is now, **who** will be allowed to make these modifications, and to what extent? This depends on the elements to be changed (e.g. logging <=> guidance) and on the role of the user, of course. As these modifications are to be made by using... **commands,** the use of these commands is once again tailorable to the role of the user.

8.5 The user profile in the IBM S/38

In the System/38, a user profile is constituted of the following parts ([S/38??], p. 526):

- Basic part: User name, special authority authorized to the user, storage (allowed and used), priority limit, initial program name, text description, number of objects owned by the user, and number of objects authorized to the user

- Commands to which the user is explicitly authorized

- Devices to which the user is explicitly authorized

- Objects to which the user is explicitly authorized and what his authority for each object is

- Objects owned by the user

In the S/38, only the security officer can create or change a user profile (using a special command).

Chapter 9: USING THE MEANS TO ACHIEVE THE ENDS

This chapter starts with showing how to build up a tailored command set and then goes on to show how the ends discussed in chapter 2 can be achieved by using the means previously discussed; this is done by taking again the different sections of chapter 2 and discussing the aspects specific to each one of them.

9.1 Building a tailored command set

To build a tailored command set means on the one hand to provide a subset or superset of all available commands and on the other hand to tailor the commands themselves, i.e. provide a subset of the operands of the commands.

Both necessitate the use of the **command editor** (as it is the only function which can act against syntax-files), the former to add commands to the user's syntax file or to delete some from it, the latter to modify given commands.

Adding or deleting a command is straightforward using the commands provided by the editor, but modifying a command in such a way that a certain "view" of the command is given needs a few words of explanation.

As has yet been shown in chapter 5, the internal interface to a command is always the same (the executing module expects given data under a certain (standard) format), while the external interface permits to define the visibility of operands at the user interface (among other things).

So, to make an operand invisible, two things are necessary: first, render it optional by giving it a default value, and second, render it invisible (there is an indicator making part of the external interface to do so: see section 5.2).

Caution is advisable: if the operand to be hidden has sub-operands, they must also be hidden and rendered optional.

Before proceeding to the ends, one important remark: I shall only discuss the aspects specific to each end, so the user profile - always necessary- will not be mentioned, but well the parts of it specific to the ends.

9.2 Improvement of initial training

First, the <u>rank beginner</u>: he has to be provided the **guided mode** of interaction and should not be permitted to modify his mode of interaction, thus he will not be provided a command to do so.

He must be assigned a syntax file containing a **subset** of all available commands; the existence of the command editor is to be hidden from him, and perhaps also the procedure and batch concepts (by hiding the corresponding commands).

The **language characteristics** discussed in chapter 3 are also very important, as, on the one hand, they help in defining the subset of commands needed by the novice, and on the other hand, they augment the Command Languages **resistance to semantic errors.** Resistance to semantic errors refers to the likelihood that a user will type something he did not mean to type, and that what he types is a valid syntactic construct nevertheless. The greater that likelihood, of course, the less the resistance ([HARD82]). Resistance to semantic errors is particularly important for novice users.

Second, the <u>advanced novice</u>: his default mode of interaction is the **unguided mode**, but he has a command at his disposition to modify this. The same structure for the different applications is guaranteed by the use of the **centralized Dialogue Manager.**

9.3 Support of evoluting user

Aside from a **tailored command set**, augmented when passing from one sophistication level to another, the most important thing is the **temporary guidance.** Yet this kind of user should also be enabled to change its guidance mode for more than just one command (i.e. by using a command).

Once again, the **language characteristics** are important because supporting the augmentation of the command set: as the user evolves toward higher sophistication levels, the language terminology must remain consistent.

These users need a way of **modifying** their **command set**; yet these modifications should not be the same for all users. They depend on the role of the users; therefore the command editor should be tailored to this role.

9.4 Plurilinguistic aspects

On the technical side, what is needed is the **command editor** to translate all names of the objects contained in the syntax files. As the Dialogue Manager is completely **data-driven**, a user can switch from one natural language to another by activating the given syntax file.

This is the end for which the **language characteristics** are the most crucial. Translating a Command Language having the size of the BS2KDO is a very tedious task. Even given the language characteristics (most importantly, consistency), there still remain 160 command names, 1000 operand names and 1200 operand values to be translated.

Discussion of the plurilinguistic aspects in a detailed way would require several chapters on its own; the interested reader should refer to [STIE85].

9.5 A standardized user interface

This is made possible mostly thanks to the **centralized, data-driven dialogue manager** with its algorithmically derived masks and sequences of masks and its **availability to application programs.**

For its use to be attractive to the application designer, the **command editor** is necessary, permitting for instance testing and prototyping of different user interfaces to the same application.

9.6 Security aspects

The **need-to-know** principle is realized by providing a **tailored command set** (corresponding to the user's role), permitting to render certain objects invisible or to disallow certain actions on given objects.

What is more, the **command editor** is the only function allowed to manipulate the syntax files; this further enhances the security aspect.

Remains the problem of **"sensitive"** or dangerous commands; there are users who must be asked confirmation before, for instance, deleting a file, while others must not (would even be bored by it).

The problem is: who has to ask this confirmation: the Dialogue Manager or the command implementor?

If it is the **Dialogue Manager**, it would always be done in a consistent way.

It would necessitate to add to the external interface an indication of the sensitivity of the command and a message to be used (as characteristic of the operation); additionally, the user profile must contain an information about the "sensitivity level" of the user, that is, its sophistication level (i.e. novice ==> expert).

Unfortunately, i-t could happen that the Dialogue Manager asks for confirmation, this confirmation is given by the user, and afterwards the implementor replies that the object doesn't exist; this would disturb the user.

If it is the **implementor**, it could happen that each implementor does it in its own way, causing consistency problems.

This time, the implementor must be provided an information on whether to ask the user or not (i.e. whether he "is sensitive" or not). This is easily done by adding an operand to the command in question, denoting the sophistication level of the user (i.e. novice to expert), and never shown at the user interface. When defining the command set for the user, this operand is given the appropriate value. In this way, the implementor is provided the information it needs.

The big advantage of this solution is that something like "OK, Delete it" "..." "Sorry, it doesn't exist" would not happen.

In my opinion, the **second solution** is the best one, because it is **neat** and totally **within the scope of** "normal" **tailoring.**

Yet it makes anew arise the question about the amount of messages the Dialogue Manager should handle (see 6.3.5). What is said is one more argument **against** letting the Dialogue Manager handle all messages.

It has been shown in this thesis that a Command Language possessing given properties, processed by a data-driven, centralized Dialogue Manager and used together with an editor for the data driving this Dialogue Manager can be used to tailor user interfaces.

In the first chapter, the most important concepts used in this dissertation were defined.

The second chapter described the ends put forward for user interface tailoring.

Chapters 3 to 8 discussed the means necessitated to achieve the ends enounced, means which appeared not to be independent one from another, but in certain cases closely tied to each other. What is more, it was seen that some of these means have interesting side-effects; for instance, the language characteristics not only augment the "user-friendliness" of a system and permit tailoring, but they also keep the Command Language from becoming too complex to be maintained.

Chapter 9 finally showed how to combine the means to achieve the ends.

Tailoring augments the "user-friendliness" of a general-purpose time-sharing system, as adapting the system to the user (statically and, in a certain measure, dynamically) makes it come closer to him. Doing this, tailoring enhances people efficiency, while it maybe does not enhance machine efficiency; this is due to the means to be used and to the characteristics of the systems under consideration.

Some of the means discussed can be enhanced to still augment "user-friendliness"; so for instance, the guidance concept can be enhanced to ameliorate user assistance in case of errors (e.g. if range error, show (or highlight) range, if semantic error, show (or highlight) help text,..) or to provide query-in-depth information (i.e. get more verbose information level by level, by repeated use of "?").

One of the main claims put forward in this thesis (if not THE main claim) was the claim for consistency of the user interface throughout the whole system. As was seen, it could not always been achieved in the system serving as the main basis.

Of course, these inconsistencies should be removed. Most of those which were pointed at had a solution proposed along with the critique.

Yet there remains an important point that is unsolved: the inconsistency on the level of the similarity between the domain concept and the application programs. Providing the user a means of entering (and quitting) an application program in the same way as a domain would render him more confortable, because it hides the "system-level" aspect.

This would means that it is up to the Dialogue Manager to call the corresponding application program when the corresponding command is issued in the control area of a given mask (similarly to a (domain) command). This way, there would also be no more need for the "statement" concept at the user interface.

Unfortunately, if this fits well in the guided mode (the implementation details laid aside), it will create another inconsistency in the unguided mode. Indeed, there must remain a means to enter this application program in unguided mode; how else than by using a... command? and there we have our vicious cercle.

Nevertheless, why not go ahead? The concepts defined throughout the thesis are certainly valid for micros. What is more, due to the fact that one is closer to the machine, it is possible to ameliorate the user interface by eliminating some restrictions specific to general-purpose time-sharing systems.

The most interesting feature to use would be the windowing technique, for error messages, help texts, the management of the mask network,...

Moreover, on a micro, it will be possible to insulate the user and to provide a one-level system, i.e. a system where a one-to-one mapping between commands and functions is possible.

One more starting point for future work is the problem of the functions reaction to input (see chapter 6), i.e. whether to extend the Dialogue Manager to handle all of it or not.

APPENDICES

Appendix A shows the enhanced typing possibility for operand values, as discussed in 5.2.2. It is taken from [WEBE84], p.234.

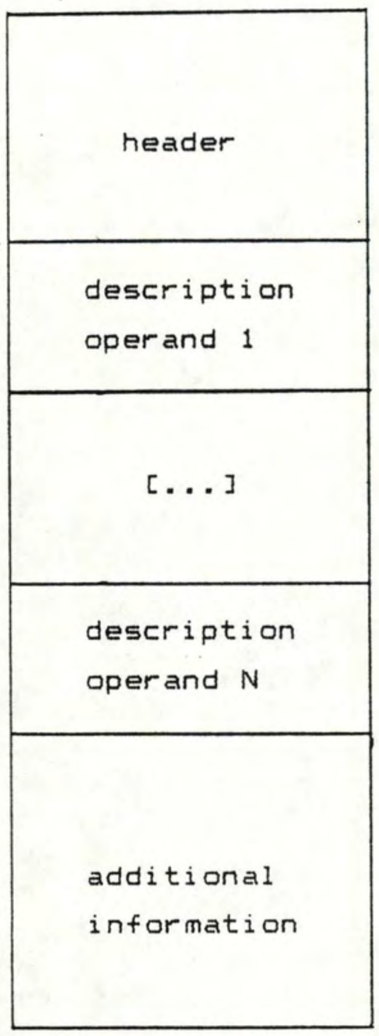
Appendix B shows and comments the standard layout form to which the external interface is mapped (see section 5.8).

Appendix C shows an example of command's characteristics as shown by the command editor (see 7.3.4).

Appendix A: enhanced typing possibility for operand values

```
ADD-VALUE
    TYPE=
      |  COMMAND-REST(SHORTEST-LENGTH=<(ANY|<integer>)>,                        |
      |               LONGEST-LENGTH=<(ANY|<integer>)>),                        |
      |                                                                         |
      |  INTEGER (LOWEST=<(ANY|<integer>)>,HIGHEST=<(ANY|<integer>)>,           |
      |          OUT-FORM=<(BINARY|PACKED|UNPACKED|CHAR)>                       |
      |                                                                         |
      |  X-STRING       (SHORTEST-LENGTH, LONGEST-LENGTH s.o.)                  |
      |                                                                         |
      |  C-STRING                          "                ,LOWER-CASE         |
      |                                                      = <(YES|NO)>       |
      |  NAME                              "                                    |
      |                                                                         |
      |  ALPHANUMERIC-NAME                 "                                    |
      |                                                                         |
      |  STRUCTURED-NAME                   "                                    |
      |                                                                         |
      |  LABEL                             "                                    |
      |                                                                         |
      |  STAR-ALPHANUM-NAME                "                                    |
      <                                                                         >
      |  KEYWORD (STAR=<(OPTIONAL| MANDATORY|FORBIDDEN>))                       |
      |                                                                         |
      |  FULL-FILENAME                                                          |
      |       (SHORTEST-LENGTH, LONGEST-LENGTH s.o., WILDCARDS=                 |
      |        <(YES|NO)>,CATALOG-ID=<(YES|NO)>,                                |
      |        <USER-ID=<(YES|NO)>,GENERATION=<(YES|NO)>,                       |
      |        VERSION=<(YES|NO)>)                                              |
      |                                                                         |
      |  PARTIAL-FILENAME (SHORTEST-LENGTH, LONGEST-LENGTH;                     |
      |                    WILDCARDS, CATALOG-ID, USER-ID s.o.)                 |
      |                                                                         |
      |  TIME                                                                   |
      |                                                                         |
      |  DATE                                                                   |
      |                                                                         |
      |  CAT-ID                                                                 |
      |                                                                         |
      |  SLASH-VALUE (SHORTEST-LENGTH, LONGEST-LENGTH s.o.)                     |
      |                                                                         |
      |  TEXT (LOWER-CASE,                 "                                    |
      |                                                                         |
      |                                                                         |
```

Appendix B: standard delivery format



<u>header</u>: contains the following information:

- length of delivery area
- internal command name
- number of operands from upper level

<u>operand description</u>: contains the following information:

- value description
- address of value or of further descriptions (case of lists of values or of structure)

<u>additional information</u>:

- values: length/"real"value
- structure descriptions
- list descriptions

structure description:

- number of operands of upper level of the structure
- operand descriptions (see above)

list description: chained description of the different values:

- value description
- address of next description
- value: length/"real"value

value description:

- type identification
- other information:
  > value is present
  > value is not present
  > value is modifiable
  > value is not modifiable
  > value is erroneous
  > value is not erroneous
  > value is to be used to replace a default value
  > value is not to be used to replace a default value

Appendix C: example of command's characteristics as shown by the
-------- command editor: command SHOW-FILE-ATTRIBUTES

```
SHØW-FILE-ATTRIBUTES

ADD-CØMMAND NAME=SHØW-FILE-ATTRIBUTES,INTERNAL-NAME=SHFAT,HELP=E(TEXT= -
            C'Gives information from the catalog entry of files'),      -
            DØMAIN=FILE,IMPLEMENTØR=P2(ENTRY=DCØFSTAT,CALL=ØLD(        -
            ØUT-CMD-NAME=FSTATUS))
     ADD-ØPERAND NAME=FILE-NAME,INTERNAL-NAME=FILENA,HELP=E(TEXT=       -
                 C'Name of the files about which information are -
requested'),DEFAULT='*ALL',RESULT-ØPERAND-NAME=*PØSITIØN(PØSITIØN=1),   -
                 CØNCATENATIØN-PØS=1
         ADD-VALUE TYPE=KEYWØRD(STAR=MANDATØRY),INTERNAL-NAME=ALL,      -
                   VALUE='*ALL'(ØUTPUT=DRØP-ØPERAND)
         ADD-VALUE TYPE=FULL-FILENAME(WILDCARDS=YES),INTERNAL-NAME=     -
                   FULLFI
         ADD-VALUE TYPE=PARTIAL-FILENAME(WILDCARDS=YES),INTERNAL-NAME=-
                   PARTFI
     ADD-ØPERAND NAME=INFØRMATIØN,INTERNAL-NAME=INFØRM,HELP=E(TEXT=     -
                 C'Amount of information requested'),DEFAULT=          -
                 'NAME-AND-SPACE',RESULT-ØPERAND-NAME=*PØSITIØN(        -
                 PØSITIØN=2),CØNCATENATIØN-PØS=1
         ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=NAMEAN,VALUE=(            -
                   'NAME-AND-SPACE'(ØUTPUT=DRØP-ØPERAND),               -
                   'SPACE-SUMMARY'(ØUTPUT='RESERVED',ØUT-TYPE=          -
                   KEYWØRD),'ALL-ATTRIBUTES'(ØUTPUT='ALL',ØUT-TYPE=     -
                   KEYWØRD))
         ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=INFØRM,STRUCTURE=YES,     -
                   VALUE='INFØRMATIØN'(NULL-ABBREVIATIØN=YES,ØUTPUT=-
                   EMPTY-STRING)
             ADD-ØPERAND NAME=STANDARD,INTERNAL-NAME=STANDA,HELP=E(     -
                         TEXT='Øutputs the access method, the VSN -
type, the last page used and the secondary allocation for the file.'),  -
                         DEFAULT='NØ',STRUCTURE-IMPLICIT=YES,           -
                         RESULT-ØPERAND-NAME=*PØSITIØN(PØSITIØN=2),     -
                         CØNCATENATIØN-PØS=2
                 ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=NØ,VALUE=(        -
                           'NØ'(ØUTPUT=DRØP-ØPERAND),'YES'(ØUTPUT=-
                           'STANDARD',ØUT-TYPE=KEYWØRD))
                 ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=STANDA,           -
                           GUIDED-ALLØWED=NØ,VALUE='STANDARD'(          -
                           ØUTPUT='STANDARD',ØUT-TYPE=KEYWØRD)
             ADD-ØPERAND NAME=PRØTECTIØN,INTERNAL-NAME=PRØTEC,HELP=E(-
                         TEXT=C'Øutputs the file security -
information.'),DEFAULT='NØ',STRUCTURE-IMPLICIT=YES,RESULT-ØPERAND-NAME=-
                         *PØSITIØN(PØSITIØN=3),CØNCATENATIØN-PØS=1
                 ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=NØ,VALUE=(        -
                           'NØ'(ØUTPUT=DRØP-ØPERAND),'YES'(ØUTPUT=-
                           'CATALØG',ØUT-TYPE=KEYWØRD))
                 ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=PRØTEC,           -
                           GUIDED-ALLØWED=NØ,VALUE='PRØTECTIØN'(        -
                           ØUTPUT='CATALØG',ØUT-TYPE=KEYWØRD)
             ADD-ØPERAND NAME=FILE,INTERNAL-NAME=FILE,HELP=E(TEXT=      -
                         C'Speciies that the FILE and the VØLUME -
informations will be produced'),DEFAULT='NØ',STRUCTURE-IMPLICIT=YES,    -
                         RESULT-ØPERAND-NAME=*PØSITIØN(PØSITIØN=4),     -
                         CØNCATENATIØN-PØS=1
                 ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=NØ,VALUE=(        -
                           'NØ'(ØUTPUT=DRØP-ØPERAND),'YES'(ØUTPUT=-
```

SHOW-FILE-ATTRIBUTES

```
                                    'TRAITS',OUT-TYPE=KEYWORD))
                     ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=FILE,            -
                                    GUIDED-ALLOWED=NO,VALUE='FILE'(OUTPUT= -
                                    'TRAITS',OUT-TYPE=KEYWORD)
                ADD-OPERAND NAME=PASSWORDS,INTERNAL-NAME=PASSWO,HELP=E( -
                               TEXT=C'Specifies whether the file is -
password protected.'),DEFAULT='NO',RESULT-OPERAND-NAME=*POSITION(         -
                               POSITION=5),CONCATENATION-POS=1
                     ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=NO,VALUE=(       -
                                    'NO'(OUTPUT=DROP-OPERAND),'YES'(OUTPUT=-
                                    'PASSWORD',OUT-TYPE=KEYWORD))
                     ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=PASSWO,          -
                                    GUIDED-ALLOWED=NO,VALUE='PASSWORDS'(   -
                                    OUTPUT='PASSWORD',OUT-TYPE=KEYWORD)
           CLOSE-STRUCTURE
      ADD-OPERAND NAME=SELECT,INTERNAL-NAME=SELECT,HELP=E(TEXT=C'The -
information must be taken from the catalog entry of the file or from -
the F1 label of the private disk.'),DEFAULT='ALL',RESULT-OPERAND-NAME= -
                    VTOC,CONCATENATION-POS=1
           ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=ALL,VALUE=('ALL'(OUTPUT=-
                          DROP-OPERAND),'BY-F1-LABEL'(OUTPUT='YES',        -
                          OUT-TYPE=KEYWORD))
           ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=BYATTR,STRUCTURE=YES(      -
                          SIZE=LARGE),VALUE='BY-ATTRIBUTES'(               -
                          NULL-ABBREVIATION=YES,OUTPUT='NO',OUT-TYPE=      -
                          KEYWORD)
                ADD-OPERAND NAME=CREATION-DATE,INTERNAL-NAME=CREATI,       -
                               HELP=E(TEXT='Selection of files by creation -
date'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,RESULT-OPERAND-NAME=CRDATE,-
                               CONCATENATION-POS=1
                     ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=ANY,VALUE=(      -
                                    'ANY'(OUTPUT=DROP-OPERAND),'TODAY'(    -
                                    OUTPUT='TODAY',OUT-TYPE=KEYWORD),      -
                                    'YESTERDAY'(OUTPUT='YESTERDAY',        -
                                    OUT-TYPE=KEYWORD))
                     ADD-VALUE TYPE=INTEGER(LOWEST=1,HIGHEST=999999,       -
                                    OUT-FORM=CHAR),INTERNAL-NAME=INTEGE
                     ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=INTERV,          -
                                    STRUCTURE=YES,VALUE='INTERVAL'(        -
                                    NULL-ABBREVIATION=YES,OUTPUT=          -
                                    EMPTY-STRING)
                          ADD-OPERAND NAME=FROM,INTERNAL-NAME=FROM,HELP=-
                                         E(TEXT='Selection of the files -
created after or at the specified date.'),DEFAULT='000101',               -
                                         RESULT-OPERAND-LEVEL=2,          -
                                         RESULT-OPERAND-NAME=*POSITION(   -
                                         POSITION=1),CONCATENATION-POS=1
                               ADD-VALUE TYPE=INTEGER(LOWEST=1,HIGHEST= -
                                         999999,OUT-FORM=CHAR),INTERNAL-NAME=-
                                         INTEGE
                               ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=       -
                                         TODAY,VALUE=('TODAY'(OUTPUT='TODAY',-
                                         OUT-TYPE=KEYWORD),'YESTERDAY'(    -
                                         OUTPUT='YESTERDAY',OUT-TYPE=      -
                                         KEYWORD))
                          ADD-OPERAND NAME=TO,INTERNAL-NAME=TO,HELP=E(  -
SHOW-FILE-ATTRIBUTES

                                   TEXT=C'Selection of the files -
created before or at the specified date'),DEFAULT='TODAY',          -
                                   RESULT-OPERAND-LEVEL=2,          -
                                   RESULT-OPERAND-NAME=*POSITION(   -
                                   POSITION=2),CONCATENATION-POS=1
                              ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=    -
                                   TODAY,VALUE=('TODAY'(OUTPUT='TODAY',-
                                   OUT-TYPE=KEYWORD),'YESTERDAY'(   -
                                   OUTPUT='YESTERDAY',OUT-TYPE=     -
                                   KEYWORD))
                              ADD-VALUE TYPE=INTEGER(LOWEST=1,HIGHEST= -
                                   999999,OUT-FORM=CHAR),INTERNAL-NAME=-
                                   INTEGE
                    CLOSE-STRUCTURE
               ADD-OPERAND NAME=EXPIRATION-DATE,INTERNAL-NAME=EXPIRA,   -
                           HELP=E(TEXT=C'Selection of the files by -
expiration date'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,                 -
                           RESULT-OPERAND-NAME=EXDATE,                  -
                           CONCATENATION-POS=1
                    ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=ANY,VALUE=(    -
                                'ANY'(OUTPUT=DROP-OPERAND),'TOMORROW'( -
                                OUTPUT='TOMORROW',OUT-TYPE=KEYWORD),    -
                                'TODAY'(OUTPUT='TODAY',OUT-TYPE=        -
                                KEYWORD),'YESTERDAY'(OUTPUT=            -
                                'YESTERDAY',OUT-TYPE=KEYWORD))
                    ADD-VALUE TYPE=INTEGER(LOWEST=1,HIGHEST=999999,     -
                                OUT-FORM=CHAR),INTERNAL-NAME=INTEGE
                    ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=INTERV,        -
                                STRUCTURE=YES,VALUE='INTERVAL'(         -
                                NULL-ABBREVIATION=YES,OUTPUT=           -
                                EMPTY-STRING)
                         ADD-OPERAND NAME=FROM,INTERNAL-NAME=FROM,HELP=-
                                   E(TEXT='Selection of the files -
created after or at the specified date.'),DEFAULT='000101',             -
                                   RESULT-OPERAND-LEVEL=2,              -
                                   RESULT-OPERAND-NAME=*POSITION(       -
                                   POSITION=1),CONCATENATION-POS=1
                              ADD-VALUE TYPE=INTEGER(LOWEST=1,HIGHEST= -
                                   999999,OUT-FORM=CHAR),INTERNAL-NAME=-
                                   INTEGE
                              ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=    -
                                   TOMORR,VALUE=('TOMORROW'(OUTPUT=    -
                                   'TOMORROW',OUT-TYPE=KEYWORD),       -
                                   'TODAY'(OUTPUT='TODAY',OUT-TYPE=    -
                                   KEYWORD),'YESTERDAY'(OUTPUT=        -
                                   'YESTERDAY',OUT-TYPE=KEYWORD))
                         ADD-OPERAND NAME=TO,INTERNAL-NAME=TO,HELP=E(  -
                                   TEXT=C'Selection of the files -
created before or at the specified date'),DEFAULT='TODAY',              -
                                   RESULT-OPERAND-LEVEL=2,              -
                                   RESULT-OPERAND-NAME=*POSITION(       -
                                   POSITION=2),CONCATENATION-POS=1
                              ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=    -
                                   TODAY,VALUE=('TODAY'(OUTPUT='TODAY',-
                                   OUT-TYPE=KEYWORD),'TOMORROW'(OUTPUT=-
                                   'TOMORROW',OUT-TYPE=KEYWORD),       -
```

SHOW-FILE-ATTRIBUTES

```
                                   'YESTERDAY'(OUTPUT='YESTERDAY',     -
                                   OUT-TYPE=KEYWORD))
                              ADD-VALUE TYPE=INTEGER(LOWEST=1,HIGHEST= -
                                   999999,OUT-FORM=CHAR),INTERNAL-NAME=-
                                   INTEGE
                    CLOSE-STRUCTURE
               ADD-OPERAND NAME=LAST-ACCESS-DATE,INTERNAL-NAME=LASTAC, -
                          HELP=E(TEXT=C'Selection of the files by -
last access date'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,               -
                           RESULT-OPERAND-NAME=LADATE,                 -
                           CONCATENATION-POS=1
                    ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=ANY,VALUE=(   -
                                'ANY'(OUTPUT=DROP-OPERAND),'TOMORROW'( -
                                OUTPUT='TOMORROW',OUT-TYPE=KEYWORD),   -
                                'TODAY'(OUTPUT='TODAY',OUT-TYPE=       -
                                KEYWORD),'YESTERDAY'(OUTPUT=           -
                                'YESTERDAY',OUT-TYPE=KEYWORD))
                    ADD-VALUE TYPE=INTEGER(LOWEST=1,HIGHEST=999999,    -
                                OUT-FORM=CHAR),INTERNAL-NAME=INTEGE
                    ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=INTERV,       -
                                STRUCTURE=YES,VALUE='INTERVAL'(        -
                                NULL-ABBREVIATION=YES,OUTPUT=          -
                                EMPTY-STRING)
                         ADD-OPERAND NAME=FROM,INTERNAL-NAME=FROM,HELP=-
                                   E(TEXT='Selection of the files -
created after or at the specified date.'),DEFAULT='000101',            -
                                   RESULT-OPERAND-LEVEL=2,             -
                                   RESULT-OPERAND-NAME=*POSITION(      -
                                   POSITION=1),CONCATENATION-POS=1
                              ADD-VALUE TYPE=INTEGER(LOWEST=1,HIGHEST= -
                                   999999,OUT-FORM=CHAR),INTERNAL-NAME=-
                                   INTEGE
                              ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=    -
                                   TOMORR,VALUE=('TOMORROW'(OUTPUT=    -
                                   'TOMORROW',OUT-TYPE=KEYWORD),       -
                                   'TODAY'(OUTPUT='TODAY',OUT-TYPE=    -
                                   KEYWORD),'YESTERDAY'(OUTPUT=        -
                                   'YESTERDAY',OUT-TYPE=KEYWORD))
                         ADD-OPERAND NAME=TO,INTERNAL-NAME=TO,HELP=E(  -
                                   TEXT=C'Selection of the files -
created before or at the specified date'),DEFAULT='TODAY',             -
                                   RESULT-OPERAND-LEVEL=2,             -
                                   RESULT-OPERAND-NAME=*POSITION(      -
                                   POSITION=2),CONCATENATION-POS=1
                              ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=    -
                                   TODAY,VALUE=('TODAY'(OUTPUT='TODAY',-
                                   OUT-TYPE=KEYWORD),'TOMORROW'(OUTPUT=-
                                   'TOMORROW',OUT-TYPE=KEYWORD),       -
                                   'YESTERDAY'(OUTPUT='YESTERDAY',     -
                                   OUT-TYPE=KEYWORD))
                              ADD-VALUE TYPE=INTEGER(LOWEST=1,HIGHEST= -
                                   999999,OUT-FORM=CHAR),INTERNAL-NAME=-
                                   INTEGE
                    CLOSE-STRUCTURE
               ADD-OPERAND NAME=SUPPORT,INTERNAL-NAME=SUPPOR,HELP=E(   -
                           TEXT=C'Selection of the files by the type -
```

SHOW-FILE-ATTRIBUTES

```
of support'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,LIST-POSSIBLE=YES(   -
                          LIMIT=3),RESULT-OPERAND-NAME=SUPPORT,         -
                          CONCATENATION-POS=1
                ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=ANY,VALUE=         -
                            'ANY'(OUTPUT=DROP-OPERAND)
                ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=PUBLIC,            -
                            LIST-ALLOWED=YES,VALUE=('PUBLIC-DISK'(  -
                            ALIAS-NAME=PUBLIC-DISC,OUTPUT='PUBLIC',-
                            OUT-TYPE=KEYWORD),'PRIVATE-DISK'(        -
                            ALIAS-NAME=PRIVATE-DISC,OUTPUT=          -
                            'PRDISC',OUT-TYPE=KEYWORD),'TAPE'(       -
                            OUTPUT='TAPE',OUT-TYPE=KEYWORD))
            ADD-OPERAND NAME=VOLUME,INTERNAL-NAME=VOLUME,HELP=E(        -
                          TEXT=C'Selection of the files contained in -
the specified volume'),DEFAULT='*ANY',STRUCTURE-IMPLICIT=YES,           -
                          RESULT-OPERAND-NAME=VOLUME,                   -
                          CONCATENATION-POS=1
                ADD-VALUE TYPE=KEYWORD(STAR=MANDATORY),                 -
                            INTERNAL-NAME=ANY,VALUE='*ANY'(OUTPUT= -
                            DROP-OPERAND)
                ADD-VALUE TYPE=ALPHANUMERIC-NAME(LONGEST-LENGTH=6),-
                            INTERNAL-NAME=ALPHAN
            ADD-OPERAND NAME=SIZE,INTERNAL-NAME=SIZE,HELP=E(TEXT=       -
                          C'Selection of the files by the number of -
pages'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,RESULT-OPERAND-NAME=SIZE, -
                          CONCATENATION-POS=1
                ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=ANY,VALUE=         -
                            'ANY'(OUTPUT=DROP-OPERAND)
                ADD-VALUE TYPE=INTEGER(LOWEST=0,HIGHEST=16777215,       -
                            OUT-FORM=CHAR),INTERNAL-NAME=INTEGE
                ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=INTERV,            -
                            STRUCTURE=YES,VALUE='INTERVAL'(          -
                            NULL-ABBREVIATION=YES,OUTPUT=            -
                            EMPTY-STRING)
                    ADD-OPERAND NAME=FROM,INTERNAL-NAME=FROM,HELP=-
                                  E(TEXT='Selection of the files with -
a number of extents greater than or equal to the specified number'),    -
                                  DEFAULT='0',RESULT-OPERAND-LEVEL=2, -
                                  RESULT-OPERAND-NAME=*POSITION(      -
                                  POSITION=1),CONCATENATION-POS=1
                        ADD-VALUE TYPE=INTEGER(LOWEST=0,HIGHEST= -
                                  16777215,OUT-FORM=CHAR),            -
                                  INTERNAL-NAME=INTEGE
                    ADD-OPERAND NAME=TO,INTERNAL-NAME=TO,HELP=E(  -
                                  TEXT=C'Selection of the files with -
a number of extents less than or equal to the specified number.'),      -
                                  DEFAULT='16777215',                 -
                                  RESULT-OPERAND-LEVEL=2,             -
                                  RESULT-OPERAND-NAME=*POSITION(      -
                                  POSITION=2),CONCATENATION-POS=1
                        ADD-VALUE TYPE=INTEGER(LOWEST=0,HIGHEST= -
                                  16777215,OUT-FORM=CHAR),            -
                                  INTERNAL-NAME=INTEGE
                CLOSE-STRUCTURE
            ADD-OPERAND NAME=NUMBER-OF-EXTENTS,INTERNAL-NAME=NUMEXT,-
                          HELP=E(TEXT=C'Selection of the files by the -
SHOW-FILE-ATTRIBUTES

number of extents occupied'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,        -
                             RESULT-OPERAND-NAME=EXTENTS,                 -
                             CONCATENATION-POS=1
                      ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=ANY,VALUE=     -
                                'ANY'(OUTPUT=DROP-OPERAND)
                      ADD-VALUE TYPE=INTEGER(LOWEST=0,HIGHEST=65535,      -
                                OUT-FORM=CHAR),INTERNAL-NAME=INTEGE
                      ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=INTERV,        -
                                STRUCTURE=YES,VALUE='INTERVAL'(           -
                                NULL-ABBREVIATION=YES,OUTPUT=             -
                                EMPTY-STRING)
                           ADD-OPERAND NAME=FROM,INTERNAL-NAME=FROM,HELP=-
                                     E(TEXT='Selection of the files -
which have a number of not used reserved pages greater than or equal -
to the specified number.'),DEFAULT='0',RESULT-OPERAND-LEVEL=2,            -
                                    RESULT-OPERAND-NAME=*POSITION(        -
                                    POSITION=1),CONCATENATION-POS=1
                                ADD-VALUE TYPE=INTEGER(LOWEST=0,HIGHEST= -
                                     65535,OUT-FORM=CHAR),INTERNAL-NAME= -
                                     INTEGE
                           ADD-OPERAND NAME=TO,INTERNAL-NAME=TO,HELP=E(  -
                                     TEXT=C'Selection of the files with -
a number of reserved pages not used less than or equal to the -
specified number.'),DEFAULT='65535',RESULT-OPERAND-LEVEL=2,               -
                                    RESULT-OPERAND-NAME=*POSITION(        -
                                    POSITION=2),CONCATENATION-POS=1
                                ADD-VALUE TYPE=INTEGER(LOWEST=0,HIGHEST= -
                                     65535,OUT-FORM=CHAR),INTERNAL-NAME= -
                                     INTEGE
                      CLOSE-STRUCTURE
                 ADD-OPERAND NAME=NUMBER-OF-FREE-PAGES,INTERNAL-NAME=      -
                             NUMFRE,HELP=E(TEXT=C'Selection of files -
with the specified number of reserved pages which are not used'),         -
                             DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,        -
                             RESULT-OPERAND-NAME=FREESIZE,                -
                             CONCATENATION-POS=1
                      ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=ANY,VALUE=     -
                                'ANY'(OUTPUT=DROP-OPERAND)
                      ADD-VALUE TYPE=INTEGER(LOWEST=0,HIGHEST=16777215,   -
                                OUT-FORM=CHAR),INTERNAL-NAME=INTEGE
                      ADD-VALUE TYPE=KEYWORD,INTERNAL-NAME=INTERV,        -
                                STRUCTURE=YES,VALUE='INTERVAL'(           -
                                NULL-ABBREVIATION=YES,OUTPUT=             -
                                EMPTY-STRING)
                           ADD-OPERAND NAME=FROM,INTERNAL-NAME=FROM,HELP=-
                                     E(TEXT='Selection of the files with -
a number of pages greather than or equal to the specified number'),       -
                                     DEFAULT='0',RESULT-OPERAND-LEVEL=2, -
                                     RESULT-OPERAND-NAME=*POSITION(       -
                                     POSITION=1),CONCATENATION-POS=1
                                ADD-VALUE TYPE=INTEGER(LOWEST=0,HIGHEST= -
                                     16777215,OUT-FORM=CHAR),             -
                                     INTERNAL-NAME=INTEGE
                           ADD-OPERAND NAME=TO,INTERNAL-NAME=TO,HELP=E(  -
                                     TEXT=C'Selection of the files with -
a number of pages less than or equal to the specified number.'),          -
```

SHØW-FILE-ATTRIBUTES

```
                                     DEFAULT='16777215',                   -
                                     RESULT-ØPERAND-LEVEL=2,               -
                                     RESULT-ØPERAND-NAME=*PØSITIØN(        -
                                     PØSITIØN=2),CØNCATENATIØN-PØS=1
                                ADD-VALUE TYPE=INTEGER(LØWEST=0,HIGHEST=   -
                                     16777215,ØUT-FØRM=CHAR),              -
                                     INTERNAL-NAME=INTEGE
                    CLØSE-STRUCTURE
                ADD-ØPERAND NAME=ACCESS,INTERNAL-NAME=ACCESS,HELP=E(       -
                            TEXT=C'Selection of the files by access -
type'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,RESULT-ØPERAND-NAME=ACCESS,-
                            CØNCATENATIØN-PØS=1
                     ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=ANY,VALUE=(      -
                                 'ANY'(ØUTPUT=DRØP-ØPERAND),'READ'(        -
                                 ØUTPUT='READ',ØUT-TYPE=KEYWØRD),          -
                                 'WRITE'(ØUTPUT='WRITE',ØUT-TYPE=          -
                                 KEYWØRD))
                ADD-ØPERAND NAME=PASSWØRD,INTERNAL-NAME=PASSWØ,HELP=E(     -
                            TEXT=C'Selection of the files which are -
protected by a type of password or which do not have any password'),      -
                            DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,          -
                            LIST-PØSSIBLE=YES(LIMIT=4),                    -
                            RESULT-ØPERAND-NAME=PASSWØRD,                  -
                            CØNCATENATIØN-PØS=1
                     ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=ANY,VALUE=       -
                                 'ANY'(ØUTPUT=DRØP-ØPERAND)
                     ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=NØNE,            -
                                 LIST-ALLØWED=YES,VALUE=('NØNE'(ØUTPUT=    -
                                 'NØNE',ØUT-TYPE=KEYWØRD),                 -
                                 'READ-PASSWØRD'(ØUTPUT='RDPASS',          -
                                 ØUT-TYPE=KEYWØRD),'WRITE-PASSWØRD'(       -
                                 ØUTPUT='WRPASS',ØUT-TYPE=KEYWØRD),        -
                                 'EXEC-PASSWØRD'(ØUTPUT='EXPASS',          -
                                 ØUT-TYPE=KEYWØRD))
                ADD-ØPERAND NAME=USER-ACCESS,INTERNAL-NAME=USERAC,HELP=    -
                            E(TEXT=C'Selection of the files which are -
shareable or not'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,                   -
                            RESULT-ØPERAND-NAME=SHARE,CØNCATENATIØN-PØS=-
                            1
                     ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=ANY,VALUE=(      -
                                 'ANY'(ØUTPUT=DRØP-ØPERAND),               -
                                 'ØWNER-ØNLY'(ØUTPUT='NØ',ØUT-TYPE=        -
                                 KEYWØRD),'ALL-USERS'(ØUTPUT='YES',        -
                                 ØUT-TYPE=KEYWØRD))
                ADD-ØPERAND NAME=STATUS,INTERNAL-NAME=STATUS,HELP=E(       -
                            TEXT=C'Selection of the files which are not -
closed'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,RESULT-ØPERAND-NAME=         -
                            STATE,CØNCATENATIØN-PØS=1
                     ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=ANY,VALUE=(      -
                                 'ANY'(ØUTPUT=DRØP-ØPERAND),               -
                                 'NØT-CLØSED'(ØUTPUT='NØCLØS',ØUT-TYPE=    -
                                 KEYWØRD))
                ADD-ØPERAND NAME=ACCESS-METHØD,INTERNAL-NAME=ACCMET,       -
                            HELP=E(TEXT=C'Selection of the files by -
access method'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,LIST-PØSSIBLE=YES(-
                            LIMIT=5),RESULT-ØPERAND-NAME=FCBTYPE,          -
```

SHØW-FILE-ATTRIBUTES

```
                    CØNCATENATIØN-PØS=1
            ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=ANY,VALUE=     -
                        'ANY'(ØUTPUT=DRØP-ØPERAND)
            ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=PAM,           -
                        LIST-ALLØWED=YES,VALUE=('PAM'(ØUTPUT=  -
                        'PAM',ØUT-TYPE=KEYWØRD),'SAM'(ØUTPUT=  -
                        'SAM',ØUT-TYPE=KEYWØRD),'ISAM'(ØUTPUT= -
                        'ISAM',ØUT-TYPE=KEYWØRD),'BTAM'(ØUTPUT=-
                        'BTAM',ØUT-TYPE=KEYWØRD),'NØNE'(ØUTPUT=-
                        'NØNE',ØUT-TYPE=KEYWØRD))
        ADD-ØPERAND NAME=BACKUP-CLASS,INTERNAL-NAME=BACKUP,HELP=-
                    E(TEXT=C'Selection of the files by the -
backup level.'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,LIST-PØSSIBLE=YES(-
                    LIMIT=5),RESULT-ØPERAND-NAME=BACKUP,         -
                    CØNCATENATIØN-PØS=1
            ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=ANY,VALUE=     -
                        'ANY'(ØUTPUT=DRØP-ØPERAND)
            ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=A,             -
                        LIST-ALLØWED=YES,VALUE=('A'(ØUTPUT='A',-
                        ØUT-TYPE=KEYWØRD),'B'(ØUTPUT='B',      -
                        ØUT-TYPE=KEYWØRD),'C'(ØUTPUT='C',      -
                        ØUT-TYPE=KEYWØRD),'D'(ØUTPUT='D',      -
                        ØUT-TYPE=KEYWØRD),'E'(ØUTPUT='E',      -
                        ØUT-TYPE=KEYWØRD))
        ADD-ØPERAND NAME=SAVED,INTERNAL-NAME=SAVED,HELP=E(TEXT= -
                    C'Selection of the files which have already -
been saved or never been saved by ARCHIVE'),DEFAULT='ANY',       -
                    STRUCTURE-IMPLICIT=YES,RESULT-ØPERAND-NAME= -
                    SAVE,CØNCATENATIØN-PØS=1
            ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=ANY,VALUE=(    -
                        'ANY'(ØUTPUT=DRØP-ØPERAND),'YES'(      -
                        ØUTPUT='YES',ØUT-TYPE=KEYWØRD),'NØ'(   -
                        ØUTPUT='NØ',ØUT-TYPE=KEYWØRD))
        ADD-ØPERAND NAME=GENERATIØNS,INTERNAL-NAME=GENERA,HELP= -
                    E(TEXT=C'Specifies if the information must -
be given for the generations'),DEFAULT='NØ',STRUCTURE-IMPLICIT=YES,   -
                    RESULT-ØPERAND-NAME=GEN,CØNCATENATIØN-PØS=1
            ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=NØ,VALUE=(     -
                        'NØ'(ØUTPUT=DRØP-ØPERAND),'YES'(ØUTPUT=-
                        'YES',ØUT-TYPE=KEYWØRD))
        ADD-ØPERAND NAME=TYPE-ØF-FILES,INTERNAL-NAME=TYPEØF,    -
                    HELP=E(TEXT=C'Informations are given only -
about file generation groups'),DEFAULT='ANY',STRUCTURE-IMPLICIT=YES,   -
                    RESULT-ØPERAND-NAME=TYPE,CØNCATENATIØN-PØS=1
            ADD-VALUE TYPE=KEYWØRD,INTERNAL-NAME=ANY,VALUE=(    -
                        'ANY'(ØUTPUT=DRØP-ØPERAND),            -
                        'FILE-GRØUP'(ØUTPUT='FGG',ØUT-TYPE=    -
                        KEYWØRD))
    CLØSE-STRUCTURE
  ADD-ØPERAND NAME=ØUTPUT,INTERNAL-NAME=ØUTPUT,HELP=E(TEXT=C'Direct -
the system-output'),DEFAULT='*SYSØUT',RESULT-ØPERAND-NAME=LIST,       -
              CØNCATENATIØN-PØS=1
      ADD-VALUE TYPE=KEYWØRD(STAR=MANDATØRY),INTERNAL-NAME=SYSØUT, -
                VALUE=('*SYSØUT'(ØUTPUT='(SYSØUT)',ØUT-TYPE=      -
                KEYWØRD),'*SYSLST'(ØUTPUT='(SYSLST)',ØUT-TYPE=    -
                KEYWØRD),'*PRINTER'(ØUTPUT='(PRINT)',ØUT-TYPE=    -
```

GLOSSARY

========

The references between brackets at the end of the concept's explanations refer to the place in the dissertation where additional information on the concept can be found.

application program: a program corresponding to a function other than an Operating System function

answer-ahead: command permitting to bypass one or several masks in a mask network (see 4.1.1 and section 4.6)

BS2000: BetriebsSystem 2000: one of Siemens's Operating Systems

BS2KDO: BS2000 KommandoSprache: the new Command Language of the BS2000

CLP: see Command Language Processor

command description: physical "container" of a command's external and internal interfaces (see section 5.4)

Command Language: a computer language to be used to "tell" the system what function it should provide (and how to provide it) (see 1.3.2)

Command Language Processor: a program "understanding" a given Command Language and calling the functions implementing the commands (see 1.4.2)

Command procedure: group (or sequence) of commands, intended to perform a given action (see 1.3.4)

Dialogue Manager: a program blending the features of a CLP and of a full screen manager (see chapter 6)

DIN NI AK 5.3.2. : Deutsches Institut für Normung e.V., Normungsausschuss Informationsverarbeitung, ArbeitsKreis 5.3.2. The DIN is responsible for standardisation in West Germany and is organized in several standardisation committees (Normungs= ausschüsse). The Normungssausschuss Informationsverarbeitung is subdivided into two Branch Committees, one of which is the FBI (Fachbereich Informationsverarbeitung - information processing), which in turn is subdivided into several sub-committees (Arbeitskreise; AK 5. = AK Programming). For further informations, the interested reader should refer to [SAUE83]. (see chapter 5)

DM: see Dialogue Manager

external command interface: part of the description of a command containing all informations causing effect on the user interface (see 5.2.1)

fill-in-the-blanks form: mask where several data of possibly different types may be entered at once (see 4.4.2)

form: see fill-in-the-blanks form

function: a given task a system can perform (see section 1.1)

guidance: process guiding the user through a computer system (see section 4.2)

implementor of a command: program (module) or procedure implementing the function underlying a command (see section 1.4)

initial context: the objects a user can act against and the actions he can perform on these objects (see section 8.1)

internal command interface: part of the description of a command containing all informations influencing the interface to the implementor and all informations influencing the internal processing of the command (see 5.3.1)

logging: memorization of command use (see section 8.3)

mask: schema represented on screen, to be used for display and input of data (see 4.1.2)

mask network: network composed of a set of masks (see section 3.6)

menu: process whereby a set of numbered choices are displayed on the screen for selection by the user (see 4.4.1)

need-to-know principle: security policy restricting information to those people who really need it to do their job and only the amount of information necessary for doing it (see section 2.7)

operand (of a command): part of a command whose value provides additional information to the function underlying the command (see 5.2.2)

OSCRL: Operating Systems Command and Response Language: a standard Command Language studied by several standardisation committees (see section 1.3)

parallel-sequential tradeoff: possibility of entering several commands in sequence (see 4.4.3)

procedure: see command procedure

responses: the messages sent to the user by the functions he uses (see 3.6.1)

role of user: the function the user has in the system (see 2.2.2)

S/38: IBM System/38: one of IBM's systems, which is workstation-oriented

SDF: System Dialog Facility: BS2000's Dialogue Manager

shell: CLP of the UNIX system

structure: syntax-element embodying several operands by putting them between brackets, expressing the logical dependency of the structure-operands (see 5.2.2)

tailoring of user interfaces: process allowing the definition of user interfaces according to the individual user requirements (see section 1.2)

temporary guidance: guidance provided just for the use of one command (see 4.4.4)

type fusion: possibility of defining different types for the same value of an operand (see 5.2.2)

user interface: a language allowing the user to control and use the functions provided by a computer system (see section 1.1)

user profile: logical concept used to match a tailored user interface to a given user (see chapter 8)

value (of an operand): actual value taken by one of a command's operands, e.g. FILE-NAME=<u>toto</u> (see 5.2.2)

BIBLIOGRAPHY

[BCS 78] : BRITISH COMPUTER SOCIETY
Working Party on Job Control Language JOD (IFIP TC 2 Working Group 2.7, Bulletin no.2, 1978), pp.68-88

[BENB81] : BENBASAT, I; DEXTER, A.S.; MASULIS, P.S. :
"An experimental Study of the Human/Computer Interface.";
CACM 24(11), 1981, pp. 752-762

[BENB84A] : BENBASAT, Izak; WAND, Yair :
"Command Abbreviation Behavior in Human-Computer Interaction.";
CACM 27(4), April 1984, pp. 376-383

[BENB84B] : BENBASAT, Izak; WAND, Yair :
"A structured approach to designing human-computer dialogues";
Int. J. Man-Machine Studies 21, 1984, pp. 105-126

[BOTT78] : BOTTERILL, J.H.; EVANS, W.O. :
"The rule-driven Control Language in System/38";
IBM S/38 Technical Development, 1978 IBM Corporation

[BOTT82] : BOTTERILL, J.H. :
"The design Rationale of the System/38 user interface";
IBM Systems Journal, vol. 21, no. 4, 1982, pp. 384-423

[BOUR78] : BOURNE, S.R. :
"The UNIX Shell";
the Bell System Technical Journal, vol. 57, no. 6, july-august 1978

[BRAN84] : BRANSCOMB, L.M.; THOMAS, J.C. :
"Ease of Use: A system design challenge";

IBM Systems Journal 23(3), 1984, pp. 224-235

[BREN80] : BRENDER, Ronald F. :
"The Case Against Ada as an APSE Command Language";
SIGPLAN NOTICES, 15(10), October 1980, pp. 27-34

[BROW82] : BROWN, James W. :
"Controlling the complexity of Menu Networks";
CACM 25(7), July 1982, pp. 412-418

[CASE82] : CASEY, Bernice E.; DASARATHY, B. :
"Modelling and Validating the Man-Machine Interface";
Software-Practice and Experience, 12, 1982, pp. 557-569

[CHRI83] : CHRISTIAN, Kare :
"The UNIX Operating System";
John Wiley and sons, 1983

[CONW78] : CONWAY, A.J.; HARVEY, D.G. :
"User-System/38 interface design consideration";
IBM S/38 Technical Development, 1978 IBM Corporation

[DATA80] : -------------
"Kurz erklärt: Benutzeroberfläche";
Data Report 15(4), 1980, pp. 43

[DAVI83] : DAVIS, Richard :
"Task analysis and user errors: a methodology for assessing interactions";
Int. J. Man-Machine Studies, 19, 1983, pp. 561-574

[DEAN82] : DEAN, M. :
"How a computer should talk to people";
IBM SYST J 21(4), 1982, pp. 424-453

[DEHN81] : DEHNIG, Waltraud; ESSIG, Heidrun; MAASS, Susanne :
"The Adaptation of Virtual Man-Computer Interfaces to User Requirements in Dialogs";
Springer-Verlag 1981

[DENN82] : DENNING, Dorothy E. :

"Cryptography and data security";
Addison-Wesley 1982

[DIN 84] : Deutsches Institut für Normung :
"Entwurf und Gestaltung von Dialogsystemen,
Teil 1 : Gestaltung von Masken";
2. Vorlage Normenentwurf DIN 66290

[DINN84] : Deutsches Institut für Normung,
Normungsausschuss Informationsverarbeitung,
Arbeitskreis 5.3.2.:
"OSCRL Standards"

[DIXO85] : DIXON, F.J. :
"Simplifying Screen Specifications - the 'Full
Screen Manager' Interface and 'Screen Form'
Generating Routines";
The Computer Journal, 28(2), 1985, pp 117-127

[ELLI80] : ELLIS, John R. :
"A LISP shell";
SIGPLAN NOTICES 15(5), May 1980, pp. 24-34

[FERN81] : FERNANDEZ, Eduardo B.; SUMMERS, Rita C.; WOOD,
Christopher :
"Database Security and Integrity";
Addison-Wesley 1981

[FRAS83] : FRASER, Christopher W. :
"A High-Level Programming and Command Language";
ACM SIGPLAN Notices, 18(6), June 1983, pp. 212-219

[GAIN81] : GAINES, B.R. :
"The technology of interaction - dialogue
programming rules";
Int. J. Man-Machine Studies 14(1), 1981, pp. 133-150

[GOOD84] : GOOD, M.D.; WHITESIDE, J.A.; WIXON, D.R.; JONES,
S.J. :
"Building a User-Derived Interface";
CACM 27(10), 1984, PP. 1032-1043

[GRAY85] : GRAY, Benson W. :
"A Methodology For Maintaining A Consistent Command Language Within A Decentralized Software Development Environment";
Digital Equipment Corporation, DEC TR-333, January 1985

[GREE79] : GREEN, T.R.G. :
"The necessity of syntax markers: two experiments with artificial languages";
Journal of Verbal Learning and Verbal Behavior, 18, pp. 481-496

[GREE84] : GREEN T.R.G.; PAYNE, S.J. :
"Organization and learnability in computer languages";
Int. J. Man-Machine Studies 21, 1984, pp. 7-18

[HARD82] : HARDY, I. Trotter :
"The Syntax of Interactive Command Languages: A Framework for Design";
Software - Practice and Experience 12, 1982, pp. 67-75

[HARV78] : HARVEY, D.G.; CONWAY, A.J. :
"Introduction to the System/38 Control Program Facility";
IBM S/38 Technical Development, 1978 IBM Corporation

[HILL83] : HILL, I.D.; MEEK, B.L. :
"The Current Programming Language Standards Scene I: The Standardisation Process";
North-Holland, Computers and Standards, 2, 1983, pp. 69-73

[HOPP84] : HOPPER, K.; NEWSTED, P.R. :
"Management Implications of Job Control Language Standardisation";
North-Holland, Computers and Standards, 3, 1984, pp. 19-27

[HOUG84] : HOUGHTON, Raymond C. :

"Online HELP Systems: a Conspectus";
CACM 27(2), February 1984, pp. 126-133

[HUCK80] : HUCKLE, B.A. :
"Designing a Command Language For Inexperienced Computer Users";
in "Command Language Directions", D. Beech, ed., North-Holland 1980

[JARD75] : JARDINE, D.A. :
"The Structure of Operating System Control Languages";
in "Command Languages", C. Unger, ed., North-Holland 1975

[JOSL81] : JOSLIN, P.H. :
"System Productivity Facility";
IBM Systems Journal, vol. 20, no. 4, 1981, pp. 388-406

[KRAN82] : KRANC, Morris E. :
"A Command Language for the Ada Environment";
ACM issue on Programming Environments, pp. 181-186

[KUGL80] : KUGLER, H.J. :
"Tools for the Construction of User Interfaces";
in "Command Language Directions", D. Beech, ed., North-Holland 1980

[LEDG80] : LEDGARD, Henry; WHITESIDE, John A.; SINGER, Andrew; SEYMOUR, William :
"The Natural Language of Interactive Systems";
CACM 23(10), October 1980, pp. 556-563

[LEDG81] : LEDGARD, Henry; SINGER, Andrew; WHITESIDE, John :
"Directions in Human Factors for Interactive Systems";
Springer-Verlag 1981

[LEVI80] : LEVINE, John :
"Why a LISP-based Command Language?";
SIGPLAN NOTICES 15(5), May 1980, pp. 49-53

[MART73] : MARTIN, James :
"Design of Man-Computer Dialogues";
Prentice-Hall Englewood Cliffs, 1973

[MORA81] : MORAN, Thomas P. :
"The Command Language Grammar: a representation for the user interface of interactive systems";
Int. J. Man-Machine Studies 15, 1981, pp. 3-50

[MOZE82] : MOZEICO, Howard :
"A Human/Computer Interface to Accomodate User Learning Stages";
CACM 25(2), February 1982, pp. 100-104

[NEWM83] : NEWMAN, I.A. :
"The Current Programming Language Standards Scene XI: Operating System Command and Response Languages";
North-Holland, Computers and Standards, 2, 1983, pp. 129-132

[NICK81] : NICKERSON, Raymond S. :
"Why interactive computer systems are sometimes not used by people who might benefit from them.";
Int. J., Man-Machine Studies 15, 1981, pp. 469-483

[NORM81] : NORMAN, Donald E. :
"The trouble with UNIX.";
Datamation Nov. 1981, pp. 140-150

[PARN69] : PARNAS, D.L. :
"On the use of transition diagrams in the design of a user interface for an interactive computer system";
Proceedings of the 24th. National Conference, ACM 1969, pp. 379-385

[PINN78] : PINNOW, K.W.; RANWEILER, J.G.; MILLER, J.F. :
"System/38 object-oriented architecture";
IBM S/38 Technical Development, 1978 IBM Corporation

[RAYN80] : RAYNER, D. :
"Designing user interfaces for friendliness";

in "Command Language Directions", D. Beech, ed., North-Holland 1980

[ROBE81] : ROBERTSON, G.; McCRACKEN, D.; NEWELL, A. :
"The ZOG approach to man-machine communication";
Int. J. Man-Machine Studies 14, 1981, pp. 461-488

[ROBI85] : ROBINSON, J.; BURNS, A. :
"A Dialogue Development System for the Design and Implementation of User Interfaces in Ada";
The Computer Journal, 28(1), 1985, pp. 22-28

[S/38??] : ----------------
"System/38 Control Program Facility - Programmers guide";
IBM ??

[SAUE83] : SAUER, W. :
"Information Processing Standardization in West Germany";
North-Holland, Computers and Standards 2 ,1983, pp. 181-184

[SCHN80] : SCHNEIDER, M.L.; WEXELBLATT, R.L.; JENDE,M.S. :
"Designing Control Languages From the User's Perspective";
in Command Language Directions, D. Beech ed., North-Holland 1980

[SCHO80] : SCHOFIELD, D.; HILLMAN, A.L.; RODGERS, J.L. :
"MM/1, a Man-Machine Interface";
Software-Practice and Experience, 10, 1980, pp. 751-763

[SILB83] : SILBERSCHATZ, Abraham; PETERSON, James L. :
"Operating System Concepts";
Addison-Wesley 1983

[SNOW84] : SNOWGRASS, Richard :
"An object-oriented Command Language";
IEEE Transactions on Software Engineering, vol. SE-9, no. 1, January 1983, pp. 1-8

[STIE83] : STIEGLER, Helmut G.; LOETZERICH, D.; SCHNEIDER, C. : "Komfortabler Mensch-Maschinen Dialog bei der weiterentwickelten BS2000-Kommandosprache"; In Software-Ergonomie, D. Balzert (ed.), Tagung 1/1983 des German Chapter of the ACM am 28. und 29.4.1983 in Nürnberg, B. G. Teubner Stuttgart

[STIE84A]: STIEGLER, Helmut G. : "Distributed User-Interface Support"; submitted to : Workshop on Operating Systems in Computer Networks

[STIE85] : STIEGLER, Helmut G.; DAHMEN, Guy : "Variants of the user interface to a general purpose Operating System based on different natural languages"; submitted to the IFIP TC 2 WG 2.7. working conference, to take place in september 1985, with subject: "The future of Command Languages: foundations for human-computer communication"

[THOM81] : THOMAS, J.C.; CARROLL, J.M. : "Human Factors in Communication"; IBM System Journal 20(2), 1981, pp. 237-263

[UNGE79] : UNGER, C.; KUGLER, H.J.; LEHMANN, N.; PUTFARKEN, P. : "Project NICOLA"; Progress report no.3, Abteilung Informatik, Universität Dortmund, Dortmund, Germany, 1979

[VAX 81] : ------------ VAX Software Handbook, Digital Equipment Corporation, 1981

[WEBE84] : WEBER, C. : "BS2KDO: Externe Schnittstellen und Funktionen: Gedächtnisverwaltung"; Computer Gesellschaft Konstanz MBH, Entwicklung Software, 1984