MASTER IN COMPUTER SCIENCE

Contribution to the design of an expert system interface

Noël, Françoise; Piette, Sophie

Award date:
1989

Awarding institution:
University of Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

CONTRIBUTION TO THE DESIGN OF AN EXPERT SYSTEM INTERFACE

Françoise Noël et Sophie Piette

Promoteur : Professeur François Bodart

Mémoire présenté en vue de l'obtention du titre de
Licencié et Maître en Informatique

Année académique 1988-1989

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

RESUME

Les dix dernières années ont vu le développement d'applications informatiques s'adressant à des utilisateurs non-spécialistes en Informatique. En particulier, de nombreuses recherches ont permis la définition de systèmes d'aide à la décision dans des domaines variés. Ces systèmes, dont certains sont appelés communément systèmes experts, sont de plus en plus performants et s'adressent à un public de plus en plus large. De ce fait, ils ont mis en évidence le besoin d'interfaces ergonomiques tant pour les personnes chargées d'introduire des connaissances humaines dans des systèmes de ce type (c'est à dire les analystes) que pour les personnes appelées à les consulter.

Ce mémoire s'inscrit dans cette perspective en ce sens qu'il présente une démarche de conception et de spécification d'une interface s'efforçant de répondre le mieux possible aux besoins d'un analyste chargé de développer des systèmes experts. Concrètement, la démarche évoquée est élaborée en référence à un système expert particulier dénommé K-Expert. Pour ce faire, ce travail se fonde sur divers éléments tant théoriques qu'empiriques. Ceux-ci nous conduisent à la définition du profil et des tâches d'un analyste type ainsi qu'à le sélection d'unités physiques de dialogue entre l'homme et la machine. Ces unités respectent un minimun de critères ergonomiques. Enfin, une architecture d'implémentation d'une application interactive telle que la construction d'un système expert est également proposée.

ABSTRACT

The last ten years correspond to a widespread diffusion of computer applications to the destination of users who are not specialists in computer science. In particular, a lot of researches have made possible the designing of systems supporting the decisional process in various domains. These systems, among which some are called expert systems, become more and more powerful and concern an increasing part of the population. By this way, they have highlighted the need of ergonomic interfaces for the people responsible for the input of human expertise in such systems (e.g. the analysts) but also for their end-users.

This dissertation is directly related to that perspective in this sense that it presents a conception and specification process for an interface which supports the needs of an anlyst busy with the development of expert systems. Concretely, the evoked process is built on the base of a particular expert system called K-Expert. To reach these aims, this work takes into account various theoretical but also empirical elements. All of them contribute to the definition of the profile and tasks of a typical analyst but also to the selection of physical units supporting the man-machine interaction. These units satisfy to a minimum set of ergonomic concerns. Finally, an implementation architecture is also proposed for interactive applications such as the building of an expert system.

ACKNOWLEDGEMENTS

We insist on thanking :

Mister François Bodart, professor at the "Institut d'Informatique" of the University of Namur (Belgium) who has consented to guide this work and who has accepted to give us many advices about its contents,

Mister Edzard de Buhr, responsible for the Artificial Intelligence Department of the enterprise ADV/ORGA at Wilhelmshaven (West Germany), who has agreed to co-ordinate the training period concluding our studies,

Mister Thomas Blümel, member of the Artificial Intelligence Department of ADV/ORGA, for the interest He has shown in this work. We intend to thank him particularly for all the "practical problems" He has helped us to solve during our stays in Germany.

We would like to thank also all the people of the THESEUS team from ADV/ORGA and the assistants of the Namur University for the support they have brought to the realization of this work.

We also wish to express our gratefulness to all the professors of the "Institut d'Informatique" for the knowledges and the education we have received during our cycle of studies.

TABLE OF CONTENTS

WARNING

The first Chapter of this dissertation has been written by Françoise Noël while the second Chapter has been realized by Sophie Piette.

The three others Chapters result from a common work.

INTRODUCTION

Since the apparition of the term "Artificial Intelligence" which has been introduced by John Mc Carthy at the Summer Research Project held at Darmouth College in 1956, many efforts have been consented in order to mimic the functionality of the human mind. The artificial intelligence which is now recognized as a computer Science discipline tries to make computers capable of showing intelligent behaviours. By intelligent behaviours, one means behaviours that would be considered as intelligent if they were observed in humans.

According to [Hols 87], "two cornerstones of intelligence are the ability to understand natural language and the ability to reason. These, in turn, represent two principal areas of research in the artificial intelligence field".

Researches about "the ability ro reason" have led scientists to conceive so-called expert systems. According to a definition proposed in [Pars 88], "an expert system is a program that relies on a body of knowledge to perform a somewhat difficult task usually performed only by a human expert. The principal power of an expert system is derived from the knowledge the system embodies rather than from search algorithms and specific reasoning methods. An expert system successfully deals with problems for which clear algorithmic solutions do not exist".

[Hols 87] has listed the adavantages of expert systems. According to him, they may :

- Provide advice during a human expert's unavailability ;
- Be easily replicated ;
- Be used simultaneously in many sites ;
- Make knowledge distribution relatively inexpensive ;
- Provide an advisory service which was unavailable in the past ;

- Free human expert's time in such a way that the latter may concentrate himself on the hardest cases. It should not be forgotten that human experts constitute a scarce resource ;

- Provide consitent and uniform advice ;

- Rely on a formalization of the knowledge manipulated by an organization and consequently, they can contribute to a better understanding of this organization ;

- Be used to preserve a particular knowledge which can disappear with a human expert ;

- Provide a basis for the training of new experts.

The development of expert systems requires not only users but also another kind of person able to capture the expert knowledge and to express it as facts and rules. Indeed, "since many experts were not used to programming computers and even if experts can program, they may not be able to access all the relevant knowledge that they have without external assistance, the expert knowledge was usually captured and encoded by an intermediary known as a knowledge engineer" [Pars 88]. This person may also be named an analyst.

Globally speaking, let's say that an expert system consists of an user interface, an inference engine and stored expertise. As a result of this, one must consider two typical man-machine interactions. The first one appears when an end-user consults the knowledges stored in an expert system shell. The second one, on its side, occurs whenever a so-called knowledge engineer introduces formalized knowledges in the considered expert system shell.

So, in order to make expert systems usable by this two kinds of users, two man-machine interfaces should be conceived in order to satisfy the end-user and knowledge engineer needs.

The term "man-machine interface" designates all the aspects of the computer applications having an influence on the user participation to computerized tasks. Among constitutive elements of interfaces, one must consider not only the physical environment by also the manners according to

which informations are introduced and consulted into the interfaced system. As a consequence of this, interface designers should try to conceive ergonomic interaction physical tools but also ergonomic softwares.

In this perspective, the aim assigned to our thesis consists of the proposal of an analyst's interface for a particular expert system shell named K-Expert. This one is developed by the German enterprise ADV/ORGA at Wilhelmshaven where we had the opportunity to accomplish the training period concluding our studies.

In order to reach this goal, we have organized this work in the following way. We begin with a presentation of some general theoritical and practical elements that should be kept in mind while conceiving a man-machine interface whatever the considered application may be.

Then, Chapter 2 presents a global analysis of the profile and of the task of the person for who we proposed an interface (e.g. the analyst).

Chapter 3 introduces main characteristics of the current version of K-Expert as also some already implemented interfaces for existing expert system shells.

Chapter 4 proposes the necessary steps to follow in order to specify the interface corresponding to an interactive application such as the building and consultation of an expert system. This chapter concludes with the presentation of the different screens we propose to implement.

Finally, in order to implement the interface proposal, Chapter 5 presents a general development architecture for interactive applications and instantiates it to our particular case while taking into account the implementation constraints that are linked to the development of the K-Expert analyst's interface.

CHAPTER 1 :

BASIC PRINCIPLES FOR THE DESIGN OF USER INTERFACES

Although the building of a "real interface design science" is in the process of development, some theories and models about the man-machine interaction are already available. Some of these theories are approximate however, it is not a serious handicap because in their current state, they can provide designers with valuable indications. Besides this theoretical background, we also have access to a large amount of empirical observations which were used to define design guidelines. Finally, taking into account the experience acquired in software development up to now , a "step by step" interface building method can be proposed.

All these elements must not be neglected because according to Norman [Norm 86], the conception of appropriate user interfaces requires two joint approaches. The first one is "to understand the fundamental principles behind human action and performance that are relevant for the development of engineering principles of design". The second one is "to device systems that are pleasant to use - the goal is neither efficiency nor ease nor power, although these are all to be desired, but rather systems that are pleasant, even fun to produce what Laurel calls pleasurable engagement" [Norm 86].

As shown in [Sca 87], this theoretical thought is fundamental because of two common beliefs (and at the same time errors) in the domain of man-machine interfaces. The first one is to consider that the only improvements for the user will arise from technological progress. And the second one consists of thinking that a limited thought about ergonomic questions is sufficient.

This chapter is structured in a way inspired by Shneiderman [Shnei 87] . It summarizes various readings. Section 1.1. presents some significant high level theories and models providing principles to take into account carefully in order to organize the design of an efficient man-machine interface. Section 1.2. offers a presentation of middle level principles which can be used to choose between design alternatives. Section 1.3. summarizes practical guidelines resulting from designers' practise As for the last Section 1.4., lists briefly recommended designing steps.

1.1. HIGH LEVEL THEORIES OR MODELS

1.1.1. Norman's model of man-machine interaction

1.1.1.1. Introduction

All the references to Norman that appearing in this Section are extracted from [Norm 86].

Generally speaking, the Norman's goal consists of the determination of the way people perform their tasks when achieving them by using a computer. To reach this goal, Norman establishes an "approximate theory" distinguishing among different stages and levels of activities. The stages of user's activities may be summarized by the following steps :

- Establishing the goal to be reached ;
- Forming an intention ;
- Specifying a corresponding action sequence ;
- Executing the action ;
- Perceiving the resulting system state ;
- Interpreting this state ;
- Evaluating the system state with respect to the initial goals and intentions.

Before giving more details about these stages and their chaining, it is interesting to "visualize" them as Norman proposes to do it. This view is shown on Figure 1.1 on the next page.

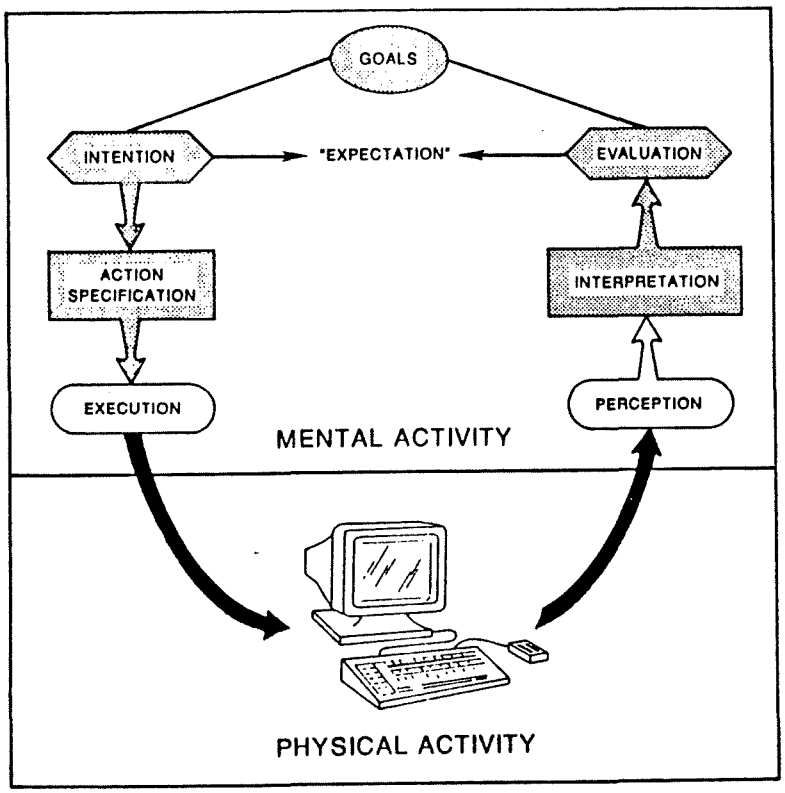


Figure 1.1 : Stages of task performance.

Starting from this, we are now speaking about the scenario of a human interaction with a computer. The way we present things is chronological. However, we must not forget that in reality, some of the steps considered can be skipped or repeated. Moreover, the chaining itself can be disturbed. Besides, a person can be "reactive" to system events rather than at the origin of these events by formulating an intention which is the situation considered here. Finally, it should be mentioned that even intentions that seem simple at a first sight (as the correction of a text via a text editor) can require the execution of various subtasks. Consequently, Norman's view of user's task is a simplification of the reality. However, his theory is particularly interesting because it highlights some crucial elements that are always present in a man-machine interaction.

1.1.1.2. Description of a typical task accomplishment

Normally, a person initiates a task by defining "goals" and "intentions". According to Norman, a goal "is the state a person wants to achieve" while an intention "is the decision to act so as to achieve the goal".

These two elements constitute the so-called "psychological variables" of the task and are directly related to a person's needs and concerns.

Since we think of the man-machine interaction problem, it seems normal to introduce now the physical system which is used to perform the work. This system can be characterized by "physical variables and mechanisms" which determine its "physical state". So, the person who wants to achieve a particular goal on a given machine must evaluate the physical state in this context. In order to realize this evaluation, the person has to translate the physical state perceived into a form compatible with the goal pursued.

The person reacts to the differences appear between the goal and the physical state by formulating an intention. To be effective, this one must be transformed into an "action sequence" which is "the specification of what physical acts will be performed upon the mechanisms of the system". The physical mechanisms are physical devices controlling the physical variables. At this point, a complex mapping from psychological goals and intentions to action sequence is necessary. Another mapping is also required. Indeed, after the realization of the action sequence the person must interpret the resulting physical state "in terms of psychological variables of interest". The last step consists of the evaluation of the system outcome by comparing the system state perceived to the initial goals. Often, this step concludes itself with the definition of new goals. These ones are treated in a manner similar to that we have just presented.

In his so-called "approximate theory", Norman emphasizes especially on the mapping operations. According to his terminology, the discrepancies at their origin can be named the " Gulf of Execution" and the "Gulf of evaluation". These notions are really crucial ones because the differences in form and contents between initial goals and physical system states are not neglegible in practise.

The two considered gulfs can be represented as follows :

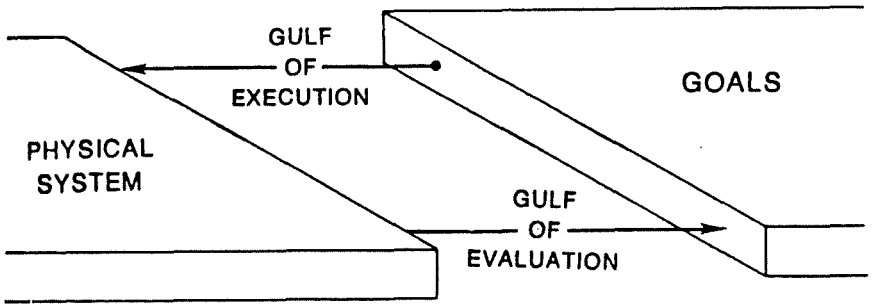


Figure 1.2 : The gulfs of execution and evaluation.

The question to ask now is related to the interest and to the importance of these mapping problems for interface design. If Norman's view of man-machine interaction is correct, it appears that the considered mappings are unavoidable. That is why it is important, whenever an interface is conceived, to try to implement it in such a way that it tends to reduce the gulfs to bridge. We are now going to see how the two gulfs can be crossed by staying at the theoretical point of view of Norman.

First of all, it is important to notice that the gulfs can be bridged by starting from the system-side or from the user-side. In the first case, the person who has to accomplish the biggest effort is the designer while the second case is more heavy for the user. Indeed,

- The designer can cross the two gulfs by bringing the system as close as possible to the user. In particular, to cross the gulf of execution, the designer must play on the input characteristics of the system in order to bring them nearer user needs. To cross the gulf of evaluation, He has just to do the same operation on the output characteristics. The result of the designer's intervention should be a system easier to use and to interpret.

- The user can cross the gulfs by himself. Globally, we can say that what He has to do is to bring the psychological elements nearer the physical ones and conversely. This requires a great deal of training and experience. In particular, to cross the gulf of execution the user must operate the translation from psychological elements to physical ones during the following four steps approach :

- Intention formation ;
- Action sequence specification ;
- Action execution ;
- Contact with the input mechanisms of the interface.

To cross the gulf of evaluation, the user must operate the translation from physical elements to psychological ones during the following four steps approach :

- Starting with the output displays of the system ;
- Moving to the perceptual processing of those displays ;
- Moving to its interpretation ;
- Moving to the evaluation.

The following Figure 1.3, on the next page, illustrates what has just been said in the previous pages of this exposure.

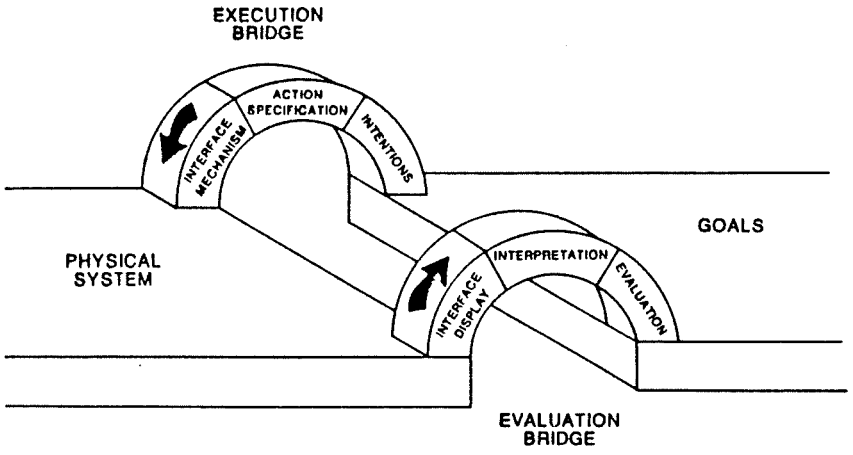


Figure 1.3 : Bridging of the evaluation and execution gulfs.

Starting from these elements, Norman gives then some practical recommendations about the manner to use them in order to move systems closer to the user. The interface supporting those systems is not obvious to conceive because the users are different from each other and moreover, "for even a single user the requirements for one stage of activity can conflict with

the requirements for another". For example, Menus can be helpful as information during intention information and action specification but they very often make the execution more difficult.

So, it seems possible to criticize interfaces according to the quality of the way they support the different activity stages. In this context,

- Reminding the user of all the abilities available can support the generation of intentions ;

- Visible items acting as a direct translation into possible actions can support the action selection ;

- Pointing devices can help considerably the management of an execution ;

- Visual reminders of what has been done should support the evaluation steps ;

- In some situations, visual structures such as graphs and pictures can be superior to text in order to facilitate the interpretation.

For Norman, the best design option would be to give the designer the responsibility to cross the gulfs in order to let the user concentrate himself on the task. This means providing a good and design model and a consistent and relevant system image. This concept of "system image" leads us to speak about the so-called conceptual models. These models provide a "scaffolding upon which to build the bridges across the gulfs". Briefly speaking, what can be said about mentals models is that they "seem a very pervasive property of humans". They are useful to help to understand an interaction , seeing that they have predictive and explanatory powers. Moreover, they can evolve through the interaction process and they can be affected by the nature of the interaction. Their main role consists of guiding human behaviours.

As far as we are concerned, we distinguish three mental models relevant for interface design. They are the Design Model, the User Model and the System Image. They are articulated between them in the following way shown on Figure 1.4 .

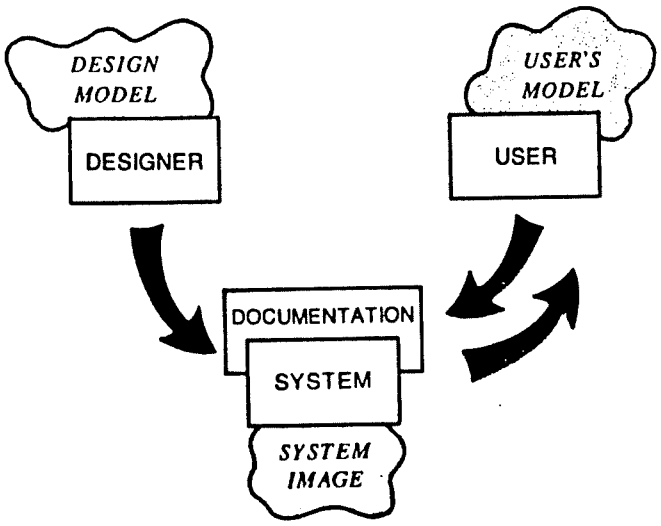


Figure 1.4 : Articulations between mental models.

They can be defined as follows :

- The Design Model is the "conceptualization of the system held by the designer". It is "the conceptual model of the system to be built" ;

- The User Model is "the conceptual model constructed by the user". "It results from the way the user interprets the system image" ;

- The System Image is "the image resulting from the physical structure that has been built (including the documentation and instructions)".

Why speaking about these three models ? Simply because they give us indications about the designer's work. Indeed, at the beginning of his work, the designer builds a Design Model. In the best case, He will take into account the user's task, requirements, abilities, background, experience, information processing mechanisms such as short-term memory limits. The User Model, however, is constructed from the system image and not from the Design Model.

So, first of all, the designer must focus himself on the elaboration of an adequate System Image. This is really important because everything that is manipulated by the user helps him to build his mental model which in turn helps him to understand what He has to do. With "everything", we mean physical knobs, dials, keyboards, displays and documentation. So, as Norman says it, "it is up to the designer to make the System Image explicit, intelligent, consistent". Note still that a compatibility between the User Model and the Design Model can only be reached through the System Image.

As example of a system in which all these models are "fine", we just have to cite the spreadsheets. In this perspective, Norman considers that the good interfaces should be perceived as tools revealing their underlying conceptual model and He puts the emphasis on the comfort, ease and pleasure to use it. Of course, this must not lead us to forget problems linked to this perception of user interfaces as tools.

Among these problems, there is the so-called "level problem". Indeed, a question to be raised is the quantity of intelligence that should be present in the considered tool interfaces. Too simple tools can cause problems because they require too much user's skills. On the other hand, too intelligent tools may also be hard to use if they do not provide some hints explaining how to use them and what they are doing at a given moment. It can be observed that tools whose components are as close as possible to the task are preferred by many people because they reduce the mapping effort to provide in order to accomplish this task. What is recommended by Norman is to provide higher level tools crafted to the task and lower level tools to modify the first ones. The retained principle is the following one : " the level of the tool has to match the level of intention".

Now that we have presented the main Norman's ideas, we can evoke some design problems related to the proposed concepts. Among these problems, we find that in practise, the number of variables and potential actions may be very extended (up to thousands !). Moreover, as a result of the numerous limitations of the current technology, the mappings discussed before are often very arbitrary.

All these proposals can be completed by taking into account the so-called feeling of "directness". We do the hypothesis that this feeling can be

expected whenever an interface is conceived in such a way that the user has to engage few cognitive resources. The directness is a relative notion and results from the interaction of many factors. Generally speaking what can be said about it is that directness possesses two facets : "distance" and "direct engagement".

By "distance", we mean the separation existing between the user's thoughts and the physical description level of the considered system. This is, as we are going to see a very important point to consider if one wants to bridge the gulfs presented previously.

By "direct engagement", we mean a qualitative feeling : the feeling to manipulate and control the tasks objects. This could also be named "a feeling of first personness". We are now going to consider these two concepts with more details.

1.1.1.3. The distance concept

This concept can be split into two points : the semantic distance and the articulatory distance which are something like linguistic properties. These two elements must be considered because each interaction with a physical device implies the use of an "interface language" in order to give a description of the actions a user intends to perform. Moreover, at the end of any interaction, a feedback is given to the user under the form of an output also expressed in an interface language. The input and output interface languages may be different but it can be a dangerous option since it stretches the explanation and evaluation gulfs.

A common point of human and computer languages is their symbolic character. Consequently, an arbitrary relation exists between a vocabulary item and its meaning. Therefore, this relation must be learned. Indeed, in general, the meaning of such an item can not be deduced directly. So, because of this separation between form and meaning, we can speak of "distance" and in particular of "semantic" and "articulatory" distances.

The Semantic Distance "reflects the relationship between the user intentions and the meaning of expressions in the interface language both for input and output".

The Articulatory Distance "reflects the relationship between the physical form of an expression and its meaning, again, both for input and output".

The interrelations between these two concepts may be resumed on Figure 1.5 in the following way :

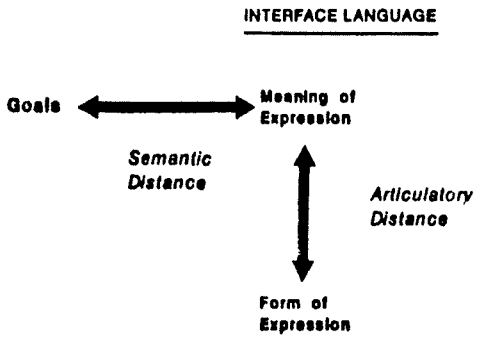


Figure 1.5 : Semantic and Articulatory Distances.

Now, we are first going to insist on the semantic distance. The "semantic directness" is a property that each interface language should possess ideally. Indeed, what is meant by "semantic directness" is the ability for the user to express everything He wants with the available interface language (e.g. : the interface language supports the user's conception of the task domain) in a very concise way (e.g. : in a straightforward way). Seeing that we have insisted in the first part of this theoretical exposure on the gulfs between the user and the system, we are now looking at the semantic distance this perspective.

1.1.1.3.1. Semantic distance and execution gulf

We have already seen that the more the considered tool level is low (like in a Turing machine for example) the more the execution gulf is large and the more the user must perform planning and translation activities. If the designed interface is semantically direct then, there is a direct correspondence between the user's way to think and the interface language description level. In this case, the volume of information processing structure that the user has to provide in order to bridge the gulf is minimized. We can see the semantic distance as an expression of "how much of the required structure is provided by the system and how much by the user".

1.1.1.3.2. Semantic distance and evaluation gulf

This time, the semantic distance expresses how much information processing structure the user has to provide in order to evaluate if the whished goal is satisfied. The more the semantic distance is small, the more the matching between output terms and user's terms is strong and the less the evaluation gulf is large.

Concretely speaking, many proposals can be done to reduce the semantic distance and, by the way, to reduce the "gulf crossing" effort.

1.1.1.3.3. From the system side

◊ Conception of higher-level input languages

These languages should be as near as possible to the user. They should enable him to express directly "frequently encountered structures of problem decomposition". The advantage is the easiness of tasks specification. However, there is a non-neglegible disadvantage : the lost of the language generality which makes the tasks difficult to decompose or even impossible to specify.

The solution that can be given to this problem consists of the conception of languages which are extensible according to everyone's needs (like in LISP or in the UNIX operating system). The price to be paid for this extensibility is an increase of the complexity and of the specialization. Indeed, "Because of the incredible variety of human intentions, the lexicon of a language that aspires to both generality of coverage and domain specific functions can grow very large".

◊ Direct image of semantic concepts by the output.

This ability is illustrated by the evolution from line-oriented text editors to screen-oriented text editors. Whatever interest this solution possesses, it does not escape to the conflict between generality and power introduced for input languages. Indeed, if the considered system is too specialized, the output lacks generality. On the contrary, a user confronted with a too rich system must face learning and using problems.

A possible solution, which is all right both for input and output languages, consists of developing special purpose systems dedicated to particular tasks. By this way, the user planning effort to bridge the gulf is reduced.

Of course, we find the price presented before again : this solution goes hand in hand with a loss of generality making things unnatural or ever impossible.

1.1.1.3.4. From the user side

◊ The user can adapt himself to the system representation

In this way to do, the user should change his perception of the tasks. This is linked to the so called "linguistic determinism". As the vocabulary seems to have an effect on our way to think things, it appears that "the interface language should provide a powerful, productive way of thinking about the domain". The problem encountered here is that if it is too much special purpose oriented, a powerful way to think about the considered task domain is available but at the same time, the risk to reduce the user's flexibility is not neglegible. Indeed, the user may be unable to think about his work by following alternative ways.

If we want that the user modifies his thinking way, and if we let him to support the gulfs bridging efforts, it seems that we must be attentive to the establishment of an equilibrium between the cost associated to the learning of a new thinking mode and the potential benefits of thinking in this way.

According to what has been said above, the bridging efforts should ideally be supported by the designer. So, if we put the effort on user's shoulders as it is done here, it should be required at least that the new model (implied by the interface) "should be coherent and consistent over some conception of the domain" in order to reduce the semantic distance as much as possible.

We conclude these considerations about semantic distance by a complementary remark. "Automated behavior does not reduce semantic distance". It does mean that even if often repeated tasks requiring cognitive efforts (like planning an action sequence) become automated, and seem less

difficult and more direct, the two gulfs must always be bridged by the user. Indeed, the only thing that actually happens in the replacement of the planning activity by a memory retrieval operation.

The Articulatory Directness is our next point of interest. First of all it can be noticed that this type of directness may be seen as the property of a language in which the physical form of vocabulary is very close to their meaning. In particular, this can be applied to interface languages. In natural language, the articulatory directness is illustrated by the onomatopoeias which introduce a non-arbitrary relation between an acoustical structure and its meaning. What should be done for interface design is to try to use technological innovations in order to implement non-arbitrary relations between physical form of inputs and outputs such as mouse movings, string of characters and the associated meaning. For example, if the task to be interfaced consists of diagrams management, it would be fine to draw them for the input and to receive diagrams on the screen as a result.

As the semantic distance, the articulatory distance can also be looked at in the context of the two predefined gulfs.

1.1.1.3.5. The articulatory distance and the evaluation and execution gulfs

The greater is the articulatory directness, the smaller is the user's effort to bridge the gulfs. So, from the input side, it appears that the best to do is to enable the user to specify an action by mimicking it (for example, moving a cursor by using a mouse). The same idea is relevant for the output side. So, a modification operated on a variable could be automatically reflected on a corresponding graphic on the screen. The principle to keep in mind may be formulated as follows : "Articulatory direct interaction can couple the interaction between action and meaning so naturally that relationships between intentions and actions and between actions and output seem straightforward and obvious". However, an important tip to give to the designer is to stay "awake", it means to get informations about the available technology because it appears that this last has a major impact on the articulatory directness.

For example, we can quote the mouse which is a so-called spatio-mimetic device. As Norman says : "That means that it can provide

articulatorally direct input for tasks that can be represented spatially. The mouse is useful for a large variety of of tasks not because of any property inherent in itself; but because we map so many kinds of relationships (even ones that are not intrinsically spatial) on to a spatial metaphor". What has been said here is linked to the wish to bring the system nearer the user. If the bridges are crossed in the other direction (e.g. : from the user), it seems possible for a user to bridge the gulfs by making the interface vocabulary items more articulatorally direct . How can He do this ? Just by changing, adapting his mental model by a reconceptualization of the existing.

1.1.1.4. The direct engagement concept

We can speak of direct engagement whenever the interface and the computer itself are transparent for the user. This last has then the feeling to work directly with interesting objects behaving as He expects. Up to now, not many things have been said about requirements to take into account in order to implement this type of engagement. What can be demanded at least is summarized by Laurel [Norm 86]. "At minimum, to produce a feeling of direct engagement the system needs :

- Execution and evaluation to be direct in the senses discussed in this chapter ;

- Input and output languages of the interface to be inter-referential, allowing an input expression to incorporate or make use of a previous output expression. This is crucial for creating the illusion that one is directly manipulating the objects of concern ;

- The system to be responsive, with no delays between execution and the results, except where those delays are appropriate for the knowledge domain itself ;

- The interface to be unobtrusive, not interfering or intruding. If the interface itself is noticed, then it stands in a third-person relationship to the objects of interest, and detracts from the directness of the engagement."

Generally speaking, what is proposed to the user is an illusion that the designer must imagine and implement as carefully and adequately as possible.

Some factors may have an impact on the preservation of this illusion. Among them, the accentuation can be placed on the form and the speed of the feedback to user's actions on objects of interest. Probably, a rapid feedback and a continual representation of the system state should be recommended because they help to make the computer transparent, they enable the user to watch actions and to monitor them like in the reality and they contribute to the minimization of the user's cognitive effort.

1.1.2. The syntactic / semantic model of user knowledge

The theoretical way to see the man-machine interactions proposed by Norman and others searchers can be completed by the syntactic/semantic model of user knowledge suggested by Shneiderman [Shnei 87].

This model completes the previous one by giving more details about the task perception at the computer side (i.e. the physical variable...etc). Indeed, the Syntactic/Semantic model classifies the user knowledges into three categories : the semantic knowledge which is composed of so-called tasks concepts and computers concepts, and the syntactic knowledge.

By referring to the Norman's model, we can link the tasks concepts to the psychological variables and the computers concepts as well as the syntactic knowledge to the physical variables. We are now going to present them briefly.

In fact, the basic principle is that users "have syntactic knowledge about device-dependent details and semantic knowledge about concepts" [Shnei 87]. These two knowledges can be characterized in the following way.

The Syntactic knowledge is :

- varied ;
- device dependent ;
- acquired by rate memorization ; and

- easily forgotten.

The Semantic knowledge is :

- structured ;
- device independent ;
- acquired by meaningful learning ;
- stable in memory because of the logical structure and of the possibility to link it to familiar concepts ; and
- hierarchically organized (from high level concepts to low level ones).

These two knowledges can be illustrated by the following Figure 1.6

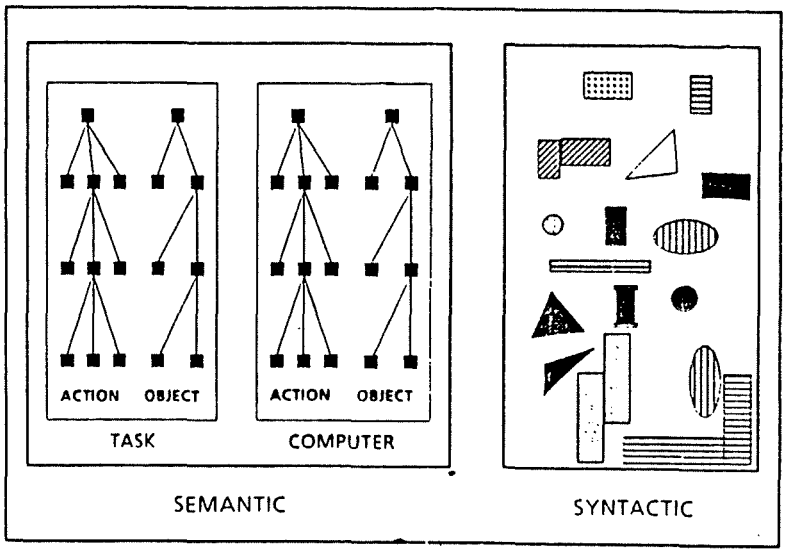


Figure 1.6 : Syntactic and Semantic knowledges.

To be more precise, we can say that the syntactic knowledge concerns the manner to follow in order to realize something with a given computer. For example, the user must remember what is the action associated with a set of function keys. There are many problems linked to this type of knowledge which should be solved in order to offer quality interfaces :

- the syntactic details are not universal ; they change arbitrarily between applications ;

- the learning process may often be associated to a struggle because of the arbitrariness that characterizes the definition of physical elements meaning (e.g. the articulatory distance of Norman). In front of this arbitrariness, the best way to acquire this syntactic knowledge is rehearsals and frequent use ;

- there is also "the difficulty of providing a hierarchical structure or even a modular structure to cope with the complexity" [Shnei 87];

- the syntactic knowledge being device dependent, a large variety can be encountered while examining several keyboards, for example. What could be recommended is a minimum overlapping of functionalities syntax across devices.

On the other hand, the semantic knowledge which is split into computer and task concepts, corresponds to the more frequent types of user's expertise. Indeed, generally, an user is more or less expert in the task realization itself and more or less expert in the "computer world".

The computer concepts consist of objects related to the computer world (such as files, directories) and of actions defined on these objects (such as saving of a file).

Generally, these concepts are conceived by highly trained computer experts. The corresponding problem is that they are not always easily accessible for novices because they can result from complex underlying software, hardware and performance constraints. The development of the people interest for computer science can perhaps contribute to the diffusion of such concepts. Indeed, the assimilation of the meaning of the RETURN key, for example, is now largely acquired.

The task concepts on the other hand are task objects (such as the notion of letter) and task actions (such as writing a letter). When one thinks about a person accomplishing a task, it appears immediately that when the treated problem is complex, the person decomposes it into smaller ones. As a

result, we can say that all the task concepts can be decomposed into more elementary ones.

With regard to the semantic knowledge, it appears from experience that designers should conceive user interfaces so that they provide examples of use, they offer general theory or pattern and they give the users the opportunity to relate proposed concepts to previous by required knowledges by analogy. The good interfaces should also describe a concrete or abstract model and indicate examples of incorrect use.

This Shneiderman's model of the user's knowledges is particularly interesting because it leads the designers to be conscious of the substantial challenge that can represent a complex interaction with a computer. Indeed, a user having only task relevant knowledges may spend some time to learn a lot of low level syntactic details before being able to realize a given task.

It leads the designers to offer user interfaces that reduce the learning efforts of computer concepts and especially of syntactic knowledge. Being aware of the existence of three knowledge kinds, the designers should systematize their design efforts. Whenever it is possible, they should concentrate themselves on the highlighting of the semantics of the task to be interfaced and design interfaces independent of specific hardware configurations. So, the user could concentrate himself on the task accomplishment and on the learning of some inevitable computer concepts (such as the "saving" concept).

Of course there exist also a lot of other high-level theories which are in full development now. As illustration of this sentence, one just has to think to the modelling of interactive system usage by transition diagrams which can be particularly "helpful" during design for instruction, and as predictors of learning time, performance time, and errors" [Shnei 87].

Another approach is the "GOMS" proposed by Card, Moran and Newell [Card 83]. By "GOMS", they mean goals, operators, methods and selection rules. The base of their theory is that users "formulate goals and subgoals that are achieved by methods or procedures for accomplishing each goal".

The operators are "elementary perceptual, motor, or cognitive acts, whose execution is necessary to change any aspect of the user's mental state or to affect the task environment" [Card 83]. Finally, we can define the selection rule as the control structure which enables a user to choose among all the methods at his disposal when He executes a task. These authors have also proposed a so-called "keystroke-level model" whose aim is "to predict performance times for error-free expert performance of tasks by summing up the time for keystroking, painting, homing, drawing, thinking, and waiting for the system to respond" [Card 83].

We are now going to conclude the presentation of some high-level theories by outlining thoughts about the typical working mode of a user that should be taken into account (even reflected) in interfaces. Indeed, it must not be forgotten that people very often do not accomplish a single task at a time but many subtasks in parallel. Remarks at this subject will lead us to evoke the multi-windowing technique.

Finally, in order to show that the theoretical studies about man-machine interactions imply the participation of various disciplines, and of psychology in particular, we intend to present some of the main ideas of the model of the "human-information processor" established by Card, Moran and Newell.

1.1.3. The Multi-windowing

The many parallel tracks of human's activities must be arranged into a single linear sequence of actions to be performed. It is what Cypher call linearization [Norm 86]. But because of the limitations of human processing resources, a lot of scheduling errors may occur during the scheduling of multiple parallel activities. If the field of ergonomics attempts to design objects that take into account the realities of the human body, the field of human-computer interaction attempts to design interfaces that take into account the realities of the human mind.

Thus, computer systems will be more comfortable for the user if they are designed so that they actively support and facilitate multiple activities. For example, computer systems should be able to provide re-orienting information in the case of external interruptions resulting from events supervening in the user's environment.

In the programs design, we believe that there will be a fairly good matching between computer programs and user activities. But in practice, there are many mismatches.

The first one is that a single activity can call upon more than one program. A system which is oblivious to the use of several programs for a single activity places the whole burden of program management on the user. When one engages in an activity, one builds up a context. But a momentary interruption will cause this context to collapse. Computer systems can provide support for interruptions by saving the complete state of a program and by assisting the user while resuming an interrupted activity. So, it can be resumed precisely where it was left off. Most window systems give the ability to trigger off multiple interactive processes for a single user. Windows are a particularly good representation for the user. They divide one screen into multiple virtual screens, each behaving like a complete screen. One of their considerable advantages is that the saved image can be presented on the screen while the user is engaged in an interrupting activity. Windows are often used to make the multiple programs of a single activity available simultaneously. Like Reichman explains it in [Norm 86], one of the most interesting aspects of windows is that they provide us with a visual display of contextualization.

The other mismatch between activities and programs occurs when more than one activity call upon a single program. Indeed, in this case, each activity wants to establish its own contents. The possibility of a context clash may be taken into account because each program has its own set of context variables.

Windows provide one way to remind users of interrupted activities. This works properly until the activities become too numerous on the screen. But also until there is one visible unit on the screen uniquely associated with each particular activity in order to remind it to the users.

Now we conclude this section by underlying some more details about windows and context management.

According to Reichman [Norm 86], it seems that most of the current window systems do not support implicit context navigation and tracking that are used by people in their everyday work. It is particularly annoying seeing

that because the contextualization supported by windows , people are inclined to assume that their daily, natural contextualization conventions are also supported by such systems. The assumption of independence between windows is an heavy one because it reduces the power and utility of window systems.

The real problem is that from one side, people see and know the interrelations linking their various subactivities while the computers do not and consequently, the latters do not possess markers to indicate such relations. So no current systems provide users with ways to indicate to the computer why they are leaving a context (e.g. a current window) and for how long they abandon it.

What window systems supporting various contexts should be able to perform is to distinguish things that must be interpreted together and things that have to be interpreted separately (e.g. for example, it leads the window system to link the windows that must be closed whenever a particular one is closed). This kind of phenomenon can be explained by the fact that some windows are functionally dependent.

Generally speaking, we can recommend the implementation and management of a so-called "controlling status" associated to each window. Indeed, as a result of what we have evoked in this Section, it appears that windows may not only be opened or closed. By reference to Reichman [Norm 86], we can say that one should consider at least the four following categories of status :

- Active ;
- Controlling ;
- Generating ;
- Closed.

Special treatments may be associated to each of them. For example, windows with a closed status should disappear from current display while it should be indicated that "controlling" windows are waiting for a return.

By the way of conclusion to this Section, let's say that what can be awaited in the near future from window systems is an accentuation of the visualization of the context interrelations. A "taxonomy of context relations" [Norm 86] (for example named arcs linking the windows and ability for the users to click on these names to return to a linked window) could be proposed. In order to not overload the user's memory, one can envisage to make visible only the windows linked to the current work on the active window. It could also be useful to reflect users the dependencies that can exist between the objects displayed within windows. Let's finish by signaling that colors, for example, may be used to show context dependencies.

1.1.4. The human information processor model

According to Card, Moran and Newell [Card 83], the human mind can be seen as an information processing system which can be described by using the same terms as for a computer. Indeed, speaking about mind we can consider processors, memories, their parameters and their interconnections. The human processor consists of three interacting subsystems called the perceptual system, the motor system and the cognitive system.

Each of them possesses its own memories and processors which are described by some parameters, as storage capacity, delay time and main code type of the information. We do not intend to go deep into the details of this model but we think it is important to underline some of its basic principles because they highlight some physical limitations of human mind that designers should not forget while conceiving an interface.

The perceptual system "carries sensations of the physical world detected by the body's sensory systems into internal representations of the mind by means of integrated sensory systems" [Card 83]. This system relies on two memories, a visual one and an acoustic one. The information is coded physically in these memories (e.g. as "an unidentified, non symbolic analogue to the external stimulus"). Shortly after the information is transmitted to a working memory where it is stored in a recognized, symbolic, acoustically or visually coded representation.

The decay of the Visual Image Store is around 200 (from 90 to 1000) msec. After this time, 50% of the information stored disappears. The decay time of the Auditory Image Store is slower. The perceptual processor

can respond after around 100 (from 50 to 200) msec. That corresponds to the response time of the visual system to a brief pulse of light.

It appears that if perceptual events occurring within a single cycle are sufficiently similar, they are combined into a single percept. What can be taken out of this for the interface design is that there is a limit to the number of acoustical and visual signals that can be presented at a time. For example, experiences have shown that 10 clicks per second can be heard where 15 clicks per second can not because of the combination of clicks that is preformed between them.

The Motor System. At the end, the thoughts are transformed into actions by activating patterns of voluntary muscles. Any movement can be decomposed into discrete micromovements whose length is around 70 (from 30 to 100) msec. The designer should especially be attentive to the arm-hand-finger and head-eye system in order to avoid to overload them.

The Cognitive System contains two types of memory, a working memory ("to hold the information under current consideration") and a long term memory ("to store knowledge for future use") [Card 83]. The short-term memory contains activated elements of the long term memory coded as acoustic or visual codes. The long-term memory consists of symbols called chunks. The notion of chunk is particularly interesting for interface design because it appears that the working memory can perceive 3 (from 2.5 to 4.1) chunks at a time. This amount can reach 7 (from 5 to 9) if the long-term memory is also used. As a result, it is the maximum number of unidimensional stimuli a man can recognize at a time. A designer should avoid to override this limit.

When a person perceives a chunk, He can associate it with another one. Consequently, when one waits that a given command name provokes a special action, it is really difficult to change this last one. This point should be kept in mind while choosing term appearing in an interface (for example when defining menu items).

As it is not relevant for our work to know much more about the different parameters and the interconnections between the three systems, we do not give an exhaustive account of our explanation. The interested reader

can consult the book written about this subject by Card, Moran and Norman [Card 83].

1.2. MIDDLE LEVEL PRINCIPLES

A striking establishment when one thinks about man-machine interface design is that designers must take into account three kinds of "intervenings". There are the people, the tasks to interface and the techniques that can be used to implement an interface.

As it is clearly underlined by Shneiderman [Shnei 87], a fundamental but universal principle could be sentenced in the following manner : "Recognize the diversity". Indeed, in order to conceive adequate interfaces, we should be aware that a particular interface can be used by various users classes, that it must give access to a well defined set of functionalities and that in this perspective, it can recourse to multiple computer techniques (such as menus, command language) which possess their own advantages but also disadvantages.

In fact, generally speaking, it could be said that interface design relies on the ability to realize the best tradeoffs it is possible in order to conciliate the three intervenings. For example, many tradeoffs must be set in order to make the designed interface as useful and attractive for all the users classes.

So now, we are going to emphasize on some characteristics of the three evoked intervenings and on the design principles which follow from them.

1.2.1. The "User" intervening

Anyone looking at people working with computers may discover rapidly that there are three big classes among them. First, there are the "novices". By reference to the syntactic/semantic model, [Section 1.1.2] it can be said that these people possess no syntactic knowledge about using the considered system and moreover, they often have only a slight semantic knowledge of computer domain. Sometimes, their task knowledge is itself not very extended and they can feel anxious about computers.

Then, we find the "Knowledgeable intermittent users". Normally, these ones master the semantic knowledge of the task and computer concepts but as a consequence of interrupted computer utilization, they face problems with regard to the memorization of the syntactic knowledge.

The "frequent users" constitute the last users group. These users master perfectly the syntactic and semantic aspects of the system. What they are looking for while using a computer interface is an efficient and rapid accomplishment of their work.

The "design principle" which follows from this is that a man-machine interface must be able to fulfil the needs of these three users classes. In other words, a first step of interface design consists of knowing the aimed users. Consequently, one can wait evolutionary interfaces because novices using frequently an interface become experts. Concretely, one can recommend a level-structured approach to learning (e.g. interfaces in which the users can progress step by step through the learning process of the syntax and of the semantic). According to Shneiderman [Shnei 87], the learning plan should be governed by the progress through the task semantics. Moreover, it can be interesting that a particular interface supports various feed-back levels (e.g. the interface returns more information to novices than to experts).

1.2.2. The "task" intervening

Knowing the users classes which are concerned by the interface design is not sufficient. Indeed, it is also necessary to know the task that must be realized by them thanks to a computer interface. A classical problem is that the task analysis is very often realized through an informal and implicit process. The main risk attached to this way to do is that the design can dictate the offered functionalities rather than the contrary (e.g. the task determines the interface).

There are two traps which threaten the designers and which are linked to an insufficient knowledge of the task [Sca 87]. There are the underfunctionality and the overfunctionality. All these two defaults may lead to the throwing out of a proposed interface. In this perspective, it seems better to offer a restricted kernel of essential functionalities than all the functionalities that can be imagined. By this way, one can hope to avoid the non-use of interfaces or a partial use of the provided functionalities or a modification of the

task itself accompanied by a strong feeling of frustration. Moreover, the task analysis is not easy to perform.

So the design principle that could be proposed from a general point of view is to know the task. From a particular point of view, as the determination of the most adequate set of atomic actions is not an obvious choice, we can propose to perform a study of the relative frequency of use before the design decisions. By this way, it should be possible to distinguish the unavoidable functionalities and to provide the easiest access to them through the future interface.

1.2.3. The "interaction styles" intervening

At the time being many interaction techniques (e.g. interaction styles) may be used in order to implement a man-machine interface. These may be classified as follows [Shnei 87]. We find :

- Menu-selection ;
- Form fill-in ;
- Command language ;
- Natural language ;
- Direct manipulation.

As we have already said it all these techniques have advantages and disadvantages. To be more precise, we should say that these two facets of each interaction style are relative to the particular users groups that can be considered. So, now, we intend to list some of the qualities and defaults of these different interaction styles.

1.2.3.1. Menu-selection

Definition :

A menu is a list of items which represent functionalities supported by the interface. The user can select one of them (with a mouse, a key-abbreviation,...etc.) in order to trigger off the associated action.

Recommended for :

Novices and intermittent users. Indeed, if the items terminology is significant users can easily perform their task by selecting the useful items which are meaningful. Seeing that meaningful character, no heavy learning and retention efforts may be provided. Moreover, well-conceived menus chaining may help users to structurate their decisions in a step by step process which guides them through the task accomplishment. This one can also support the learning of the performance of a task with a computer. Finally, it can likewise be remarked that they reduce the key strokes and by the same time, they minimize the typing errors.

Defaults :

Bad organized menus chaining may loose users. Moreover, frequent users do not always like them very much. Indeed, they require the user leaves the keyboard to move a mouse. However, this disadvantage should be alleviated by the implementation of abbreviations which provide a direct access to an item functionality. Finally, from a technical point of view, let's note that they consume an important screen space and that they require computer systems offering a rapid display rate.

Remarks :

There exist various kinds of menus (such as single menu,pop-ups...etc). However, we do not want to detail them in this work. The interested reader may have a look on [Shnei 87 Chapter3].

1.2.3.2. Form fill-in

Definition :

A form fill-in is a set of related text fields displayed on the screen at a time to enable an user to enter semantically linked data. For example, this can be useful when one needs to introduce all the data related to a book in a library management computer system.

Recommended for :

Advanced intermittent users and experts. Indeed, these forms simplify the data entry but they require a minimum training. It is necessary to be able to manipulate the keyboard, to switch between fields, to know their meaning and their syntax...etc. Consequently, an on-line help system should be implemented to support the filling of a form.

Defaults :

The most obvious default is that the display of a form on a screen may necessitate a lot of space on it.

1.2.3.3. Command language

Definition :

A command language is a language that is understandable by a computer system. It has got its own syntax and semantic. It admits a well-defined vocabulary that must be learned by users.

Recommended for :

Frequent users. Indeed, this kind of interaction style seems to go hand in hand with control and initiative abilities. Without being distracted by screen prompts (such as menu display) and without having to leave the keyboard, users may initiate system actions. By this way, users are not confined into a predefined process (like in menu chaining), they are free and command the system. This characteristic may grow their satisfaction of use. Moreover, a command language may be particularly efficient seeing that users can define so-called "macro-commands" by which they can initiate a complex actions sequence in response to a single command line.

Defaults :

They are only envisageable for expert users because they can be forgotten easily. They are coupled with a learning and memorization problem. As their syntax may be arbitrary, it appears that they put the gulf bridging effort, evoked in previous pages, on the user's shoulders. Moreover, as they rely on the keyboard use, they can be linked to a high rate of typing errors.

1.2.3.4. Natural language

Definition :

Natural language is the everyday language of people. Let's note immediately that this interaction style is submitted to many researches at the time being but that up to now, it is not the most efficient one nor the most successful.

Recommended for :

Users knowledgeable about a task domain which is very limited or intermittent users who are unable to memorize a complex command language. The biggest advantage is that there should be no syntax learning effort to provide.

Defaults :

Up to now, the researches and limited implementations like in the GURU expert system shell have not led to really convincing results. Moreover, the typing effort may be non negligible (e.g. one may have to realize more keystrokes than while using another interaction style). There is also the problem of unpredictability and of the impossibility to show the context for issuing the next command.

1.2.3.5. Direct manipulation

Definition :

Direct manipulation is a term introduced by Shneiderman to qualify interfaces possessing the following properties [Shnei 87] :

- Continuous representation of the object s of interest ;
- Physical actions or labeled button presses instead of complex syntax ;
- Rapid incremental reversible operations whose impact on the objects of interest is immediately visible.

So, concretely, direct manipulation may be seen as the putting into practise of the "what you see is what you get" principle. The direct manipulation interaction style is notably illustrated by the spreadsheets, the display editors and the video games. As it is underlined by Norman [Norm 86], the promise of

Direct Manipulation is that instead of an abstract computational medium, all the programming is done graphically, in a form that matches the way one thinks about the problem.The desired operations are done simply by moving the appropriate icons onto the screen and connecting them together.

Recommended for :

The three kinds of users. Indeed, novices can be helped because there is no heavy learning effort (e.g. often a demonstration of use is sufficient) and moreover the exploration of the system abilities is encouraged. There is an immediate visual reaction to each of their actions so they are not lost. Moreover, the error level may be reduced to a minimum level and consequently error messages are rarely needed. The users' anxiety should be reduced because "the system is comprehensible and because actions are so easily reversible" [Shnei 87]. In this case, the gulf bridging efforts are supported by the designers, the knowledges about semantic and syntactic computer concepts are reduced to a minimum in such a way that the users may concentrate on the task-related knowledges.

Intermittent users are interested because the efforts of retention over time are reduced. They can keep in mind the fundamental operational concepts.

Experts should have the opportunity to accomplish their task very quickly. However, this is criticized by Norman [Norm 86]. For him, the speed at execution is not a relevant factor because it seems that the use of an interaction style such as command language could be more efficient. Generally speaking, it could be said that for the three users categories, a high satisfaction level can be awaited and also the development of interfaces supporting a so-called "direct engagement" feeling.

Defaults :

As a counterpart to the important qualities of well-designed direct manipulation oriented interfaces, many problems emerge :

- Such interfaces may be difficult to implement. Indeed, the direct manipulation requires that one finds good representations of the task objects. Moreover, this kind of interaction style does not fit to each application kind ;

- They rely on graphics display and on the use of pointing devices ;

- Another problem may be found in the fact that direct manipulation tries to support directly the way users think about a task domain. The consequence of this is that such an interaction style may inhibit the providing of new ways to think of and to interact with a task domain. Consequently, this can block users in the existing and prevent designers to exploit what Norman calls the most exciting potential of new technology [Norm 86] ;

- Direct manipulation is not a solution to an insufficient understanding of the task domain ;

- Finally, there is the problem of error occurrence. Indeed, Shneiderman assumes that no error message is necessary because every action is immediately reflected on the screen. However, there is a linked problem in this sense that errors have to be made visually obvious. Such an option may require many design efforts.

1.3. PRACTICAL GUIDELINES

High level theories and analyses of the user of the tasks to interface and of the most suitable interaction styles are the first steps of interface design. However, for more general aspects of interface design, it is possible to distinguish some basic ergonomic principles that should be considered while conceiving well-designed interfaces.

These ergonomic guidelines may be formulated by different ways. They can be sentenced as "golden rules" as it is done by Shneiderman [Shnei 87]. But many books and papers are written about the subject. They give advices about the way to organize screen displays (input and output) and also the dialogue (e.g. the communication between man and machine). Among many others, we can consult [Sca 87], [Shnei 87],.[Fau 82],.[Cout 87],.[Norm 86],.[Brown 88].

Globally speaking, we can classify ergonomic concerns in the following categories inspired by [Sca 87] :

- Compatibility :
  Compatibility of the screen contents, users's vocabulary and way to perceive tasks-related concepts ;

- Consistency :
  It means that the same commands sequences are used to reach the same results, that the same functionalities are designated by the same terminology through the whole interface. This principle seems to be "the most frequently violated one, and yet the easiest one to repair and avoid" [Shnei 87] ;

- Concision :
  The human information processor model (presented in Section 1.4.) has underlined the existence of limits in the human short-term memory. So, ergonomic interfaces should alleviate it by not imposing memorization of complex commands sequences to users. According to [Sca 87], the computer must be considered as an external memory. In the same context, let's remark that it can be interesting to minimize the number of actions to perform in order to accomplish a given subtask as also the computer response time ;

- Flexibility :
  Two kinds of flexibility may be considered. Internal flexibility and external flexibility. Indeed, it can be required that best designed interfaces be able to cope with the three various users classes (e.g. offering shortcuts to experts). This corresponds to the internal flexibility. The external flexibility on its side, implies the ability for a given interface to run on various computer environments;

- Feed-back and guiding :
  Feed-back to actions performed by users, seems particularly useful because it appears that the knowledge of the action result may have an impact on the performance quality. Consequently, one can recommend a feed-back as immediate and explicit as possible. This point justifies the interest of the direct manipulation interaction style ;

- Explicit control :
  Best interfaces appear to be driven by users (rather than the contrary). As a result, it is recommended that at any time, the user has the ability to

know which processing state has been reached. Surprising systems should be avoided as much as possible. This principle may be perceived as an effort to avoid the acausality. Moreover, it is important that actions sequences possess clearly defined beginning, middle and end. So that their completion goes hand in hand with a feeling of accomplishment and that the user can prepare the next actions sequence ;

- Error handling :
  The basic idea is to prevent error occurrence (for example by using particular interaction styles rather than as other one). Errors should be reduced to non-serious ones and every potential destructive action should be confirmed explicitly by the user. If despite this prevention effort, some errors occur, they must be detected immediately by the system. This one has to signal them to the user and to assist him explicitly during the error recovery step. In particular, it can be noted that good interfaces should not oblige users to type again completely an erroneous text field. There should be the ability to correct directly the error from the already filled field.

As a result of the various ideas evoked here, we are now concluding this first chapter by a rapid list of steps that seem to be "unavoidable" ones for interface designers.

1.4. RECOMMENDED DESIGNING STEPS

As a consequence to all the things we have said, it appears that the designing process is a dynamic one. Moreover, it can require a wide time period and it can necessitate the intervention of a lot of people (such as ergonoms, programmers, task analysts and of course end-users). Briefly speaking, it can be said that at least the following enumerated stages are necessary.

1.4.1. Guidelines establishment

This step corresponds to a collect of current performances starting from already existing similar system, from the needs of all the implied parties

(e.g. managers and end-users). Some working guidelines should be gathered about subjects like the used character set, the accepted response times...etc.

1.4.2. Participatory design

We have insisted about the importance of the user and task knowledge for interface design. Offering users the opportunity to express their point of view may be really interesting in order to know exactly what they need, to determine their computer knowledges and capacities but also to reinforce a sort of feeling of ego involvement. However, one should not forget that this way to proceed is costly, time-consuming and can be at the origin of conflicts between the users who are involved and the other ones.

1.4.3. Pilot studies

Pilot studies intend to collect users actions to first screen proposals as soon as possible. They present the advantage of being inexpensive as they can be realized even by using type-written versions of envisaged screens. Moreover, they are rapid and can be very productive in order to give an adequate orientation to the conceived interface from the early beginning. They give also the opportunity to test alternatives.

1.4.4. Rapid prototype system

As it often happens with software, when a product is completely implemented its maintenance and adaptation to real users needs may be particularly costly. So, it is recommended to implement significant prototypes to offer users a realistic impression about the final look of the considered interface. Moreover, looking at the manner users work with a prototype, designers can be led to modify their way to organize the functionalities supported by the interface.

1.4.5. Acceptance tests

When the final interface is totally implemented at the conclusion of the other steps, one has still to control its quality by reference to measurable criteria such as :

- The time to learn specific functions ;
- The speed of task performance ;
- The rate of errors ;
- The user subjective satisfaction ;
- The human retention of commands over time. [Shnei 87]

These tests should be realized on significant users classes and their extent may be relative to the interface size.

The global aim attached to the recourse to these stages is to avoid time and money wasting by adapting the designed interface as soon as possible.

CHAPTER 2 :

ANALYST'S PROFILE AND TASK

In order to conceive a "good" interface for an environment supporting the building of an expert system, the profile of its future user should be defined as precisely as possible to fulfil the "know the user" principle presented by Shneiderman [Shnei 87]. Moreover, it is necessary to be conscious of the nature of the tasks the analyst performs while instantiating an expert system shell. This is a response to the Shneiderman's "know the task" principle. So, Section 2.1. defines the user profile general characteristics while Section 2.2. presents an analysis of the different subtasks to take into account. The aim of this chapter is thus to highlight the moments when a tool can be helpful for an analyst and the type of functionalities that such a tool should support.

2.1. DEFINITION OF THE USER PROFILE : THE KNOWLEDGE ENGINEER

First of all, we need to precise the meaning of the words "Knowledge Engineer". This person who can also be called the "analyst" or the "developer", is in fact responsible for the instantiation of an expert system shell with particular expert's knowledges. He must also take into account the profile of the end-user of the system in order to realize a parametrization of the developed instantiation which is relevant to the needs and abilities of this person.

From a technical point-of-view, the analyst should master the basic mechanisms of expert systems functioning. He should foresee the effects of the use of the different authorized chaining modes and be competent to represent knowledges in the formalism offered by the considered expert system shell (rules, frames ...etc).

Nevertheless, his computer knowledges have not to be very extensive, He is not necessary a specialist in computer science.

So, we can say that the analyst's intervention is located between experts who are specialists in a particular domain and users having recourse to his implementation of the extracted knowledges. Seeing the intermediary character of this position, the analyst must be able to interact with these two categories of people.

Concretely speaking, He has to be sufficiently qualified for interviewing the experts in order to extract their knowledges and for formalizing these knowledges according to an format adequate for the shell. So, it does not seem exaggerated to require communication and abstraction abilities from the analyst.

Moreover, the analyst should also have a good understanding of all the users classes and of the tasks that these ones will have to realize with the given expert system. He must help the users to define exactly what the artificial expert should do in order to replace an absent human expert efficiently. In other words, He has to think about the kind of problem a user submits generally to a human expert in the considered context.

Moreover, the analyst has to test his work with the experts in order to control the correctness of his formalization of their knowledges and then with each users class to verify that what He has implemented is really relevant and adequate for them. The analyst should be able to accept and take into account numerous criticisms.

Finally, He may also be in contact with a third person. Indeed, the developed expert system must very often be integrated into an other application or have access to external procedures in order to perform consultations. We have said that the analyst is not always competent in computer science, so, in order to reach the best integration it is possible, He can also have to work with programmers.

2.2. ANALYSIS OF THE TASK

The basic task of a knowledge engineer consists of the building of an expert system. First, we are going to propose a process in 6 steps that might be followed by an analyst when He decides to build a particular expert system [Mou 89]. Facing this method, we shall be able to determine some of the features that an efficient expert system shell environment should offer ideally.

2.2.1. The 6 phases of the analyst's task

◊ Phase 1 : Selection of the appropriate problem

During this phase, the analyst defines the domain and the task that the planned knowledge system is waited to perform competently. By the analysis of the features of the project, He will decide if the expert system technology is suitable for the considered problem. Indeed, the considered task must be appropriate and within the reasoning scope of an expert system. He must also estimate if the needed expertise is available.

If the expert system is foreseen to be only part of another software application, this phase is also very important to determine exactly which part of the application will be taken into account by the expert system and how to proceed to allow an easy integration with other software components in the future.

◊ Phase 2 : Development of a prototype system

A prototype is a significant part of the application. It is a subset of significant knowledges needed to realize some of the expert's activities.

This prototype must be realized for a small cost and in a short time. However, it plays a prominent role in the development of a knowledge system.

During this phase, the engineer learns the vocabulary and the notions related to the domain of the application. On the other side, the expert learns to formalize his knowledges and to explain his reasoning strategies.

The implementation of a prototype is a crucial step because it helps the analyst to estimate the required investment, the most appropriate structuration of the knowledges, the availability of the experts and the power of the tool used to implement the prototype.

After completing a knowledge system prototype, the analyst has to test it with care and to observe the strengths and weaknesses of its performance. It is possible to review basic design decisions because a prototype is small and can be changed radically or even discarded for only a small cost.

At the end of this point, the analyst is able to choose an expert system shell which will fit the best the chosen representation of the knowledges and strategy controls.

◊ Phase 3 : Development of the complete system

First, the "kernel" of the expert system has to be implemented. Then, the analyst will test it with the experts and, step by step, after studying new cases, He will add new knowledges. New cases will also help to test the robustness of the expert system reasoning.

The "kernel" could be the prototype (if the design decisions must not be changed), but the engineer can also start without any formalized knowledges. He can build a completely new "kernel" in respect with the choices made by reference to the prototype.

The process of acquiring knowledge, encoding it, and reviewing the knowledge system performances continues until retained predefined performance criteria have been satisfied.

During this phase, the analyst must also develop the end-user interfaces and test them with the different corresponding classes of users.

◊ **Phase 4 : Evaluation of the system**

The engineer must determine whether the system provides good performances. The best way to test it consists of putting in competition the system and human experts for the solution of a set of typical cases. Then, the results should be compared. The system is expected to be able to solve a majority of cases.

◊ **Phase 5 : Integration of the system**

During this phase, the system is interfaced with Data Bases and other applications which have been developed concurrently in parallel by using a classical analysis method.

◊ **Phase 6 : Maintenance**

Like all other software applications, an expert system can be modified in the future. One waits that the chosen tool offers help abilities to change the knowledges easily while preserving the consistency of the whole knowledge base.

2.2.2. Deduced features for an expert system shell environment

By reference to the previous section, it appears that the considered analyst's environment consists of a set of tools that should be helpful during the development of a prototype system, the development of the complete system, the evolution and the maintenance of the instantiated expert system. The integration phase is not really dependent on the environment tools but rather on the power of the expert system shell itself. Consequently, the latter has to offer integration opportunities.

Facing these considerations, we can deduce that the building environment must be a tool helping the analyst to develop a prototype easily, quickly and at a low cost. So in order to fulfill these requirements, the tool should be available on P.C.'s or on workstations. However, the tool must also be powerful enough to support the whole expert system implementation. As the development of a knowledge system is a process relying on a step by step construction of the knowledge base, the tool should offer basic features such as :

- Features linked to the building and the updating of an instantiated expert system such as an editor to create and update the knowledge base, a module listing all or only some parts of the knowledges, and a module helping to preserve the consistency of the knowledge base ;

- Features related to the testing of an instantiated expert system (an inference engine offering different control strategies, a module supporting the debugging of the knowledge base during execution) ;

- Features making possible the adaptation of an instantiated expert system to particulars users classes.

What should be kept in mind at the end of this chapter is that a good interface expert system shell should offer at least these general functionalities and present them in a manner which is compatible with the main characteristics of the analyst's profile.

CHAPTER 3 :

OVERVIEW OF EXPERT SYSTEM INTERFACES

The theoretical part of our work has insisted on the definition of an adequate set of functionalities enabling an analyst to accomplish efficiently and easily a given task. Thus, we dispose of the critical bases for the building of K-Expert interfaces.

However, we think that it can be time-sparing and source of imagination to outline a brief "state-of-the art" of some existing expert system shell interfaces under the light of our theoretical considerations. We hope by this way to extract interesting and efficient functionalities and also man-machine interaction methods.

Thus, the main underlying aim of this chapter is to present and to criticize some already existing expert system shells in order to have in mind the characteristics of their analyst's and user's interfaces and the functionalities that can be accessed through these interfaces.

The steps that lead our criticism consider separately the knowledge engineering and the user (or run time) interfaces. We focus ourselves especially on the analyst's interface. So, we first summarize set of functionalities enabling an analyst to accomplish efficiently and easily a given task. Then, we present the basic interaction "techniques". After this, we formulate some remarks about the initiative and control feelings, the internal and external flexibility, the feedback, the short and long term memory load. We also think about the errors handling, the on-line help, the dialogue interruption abilities (such as canceling, deleting, starting again, finishing) and finally about the interface consistency.

Ideally, we should have met many analysts and users to summarize their reactions to different Expert System shells. For practical reasons, we have not had the opportunity to do that, so we have limited ourselves to an analysis net based on theoretical elements. At any time, we have tried to set our opinions by "playing" ourselves with the criticized shells.

Thinking to the nature and characteristics of K-Expert, we have decided to focus ourselves on comparable shells (e.g. same levels shells). These shells present many features of what Harmon calls "the Mid-sized" expert systems tools. This author characterizes them by the following elements [Har 86] :

- They allow a serious programmer to develop and field a Mid-sized expert system on a PC ;
- They are able to draw data from external Data Bases ;
- They are able to rely on external programs ;
- They can include procedural code within a program ;
- They allow the user to partition the rule base ;
- They take advantage of multiple instantiation and simple inheritance ;
- They implement confidence factors or Baysian probabilities or make it possible for the analyst to easily encode them ;
- They make easy to create any desired user interface display ;
- They make easy the use of graphic displays ;
- They are supported by documentation ;
- The companies which trade them offer training programs to teach their use.

Among these Mid-sized tools, we have selected M1 (Section 3.2.) and Nexpert (Section 3.3.).

We conclude this chapter by highlighting interaction methods and functionalities that we think interesting to retrieve for the building of K-Expert interfaces(Section 3.4). Our criterion for "interest" is the joint presence of a theoretical justification and of the possibility of a practical implementation. The highlighting of functionalities should provide K-Expert developers with more ideas for the future evolution of the software.

But first, let's give a general presentation of K-Expert (Section 3.1.).

3.1. K-EXPERT

3.1.1. General presentation

K-Expert is a rule-based expert system shell, implemented in C language and designed to run on various IBM Pc and compatibles.

The aim of this tool consists of developing expert systems that can be integrated easily in conventional software applications. So, it can be said that K-Expert tends to be as "opened" as possible.

In this tool, the facts are represented as simple attribute-value pairs and the relationships between them are embodied by "IF-THEN" production rules. The conditions of the "IF" part of a rule (Antecedent) are connected with the boolean operators "AND", "OR" and "NOT". At the time being, no certainty factors are usable. Moreover, the knowledges being stored in external Data Bases, their size can nearly be as great as desired. The supported control strategies are the backward chaining, the forward chaining and a combination of the two previous modes. Breadth-first and depth-first are also considered.

In the future, other rule formalisms should likewise be envisageable such as frames, nets and graphs and in a further future, it could also be possible that "hyper media" (e.g. : nets of tools) be proposed.

The designers of K-Expert want an expert-system shell really easy to use so that the analysts (and obviously the end-users) must not be experts in programming and computer science. Moreover, they concentrate their efforts in order to make K-Expert portable as much as possible. This portability relies especially on the fact that K-Expert is implemented in C language. This is particularly interesting seeing that many other languages have interfaces to C and that the contrary is also right. So it should be possible to link programs written in different languages with K-Expert.

These two concerns are illustrated by the two development directions pursued at the time being :

- On one side, the designers put the accentuation on the "analyst's interface" of the K-Expert; they wish that an analyst can use K-Expert as a simple tool which will perhaps be extended so that it will be possible to access spreadsheets or other expert systems from a given one.

- On the other side, the designers intend to touch the professional programmers by giving them the ability to use K-Expert as a library of functions. In this perspective, the comfort of use is not so important than in the previously evoked objective but what is especially necessary is the availability a set of functionalities which should be as large as possible..

3.1.2. The existing analyst's interface

At the time beeing, to build the knowledge base, the analyst can use either an external text editor or the so-called "Knowledge Engineering Interface".

The first one is a classical ASCII text editor. It gives the analyst the advantage of working with a familiar and powerful editing environment for writing rules. We think however that it is not a really suitable manner for testing a knowledge base. Indeed, each time the analyst wants to change something inside it, He has to leave the K-Expert environment in order to enter the used editor. Another problem is that before being able to work on the created rules set with the K-Expert inference engine, the analyst must previously run a specialized program named "Rules Parser". This application verifies rules and variables syntax and adapt them to formats compatible with K-Expert.

The second ability consists of the use of the "Knowledge Engineering Interface". This one is developed with the so-called "Knowledge-Man" software. This is not an ideal way to proceed because the analyst must have the whole K-Man environment at his disposal in order to run the K-Expert interface itself. This one offers a basic rules editor, a variables editor and a run-time environment illustrated in Appendix A. The rules editor considers only the "AND" operator in the rules antecedent while the consequent may consists of only one action. The variables editor, on its side, allows the analyst to associate questions to variables. These questions will be displayed during a consultation if the inference engine needs some more informations to instantiate a particular variable.

When the analyst wishes to test the quality of an instantiated expert system, He has to enter the run-time environment. If needed, He can introduce some facts and has to choose the control strategy before starting the consultation. In the current system, this run-time environment is the same for the analyst and for the end-user. Like the two previously evoked editors, this tool is an elementary one from an ergonomic point of view.

During a consultation, one has access to a particular window splited into three parts. This window is illustrated by the Figure A.1 of Appendix A. The first one displays the dialogue itself. That is in this area that the inference engine questions the user to obtain informations. As feedback, it displays also the user's answers. Moreover, each time a question appears on the screen, a list of all the possible answers generated by the system is displayed in the second part of the window.So, the user has the ability to answer system questions either by typing on the keyboard or by performing a mouse selection in the list. To be complete, let's say that there is also another run-time opportunity which consists of asking "WHY" a question is generated by the inference engine. In this case, the system displays the current tried rule in the third window area. This space is also used to show the user which conclusions have been reached up to now by the inference engine. To have access to more details about K-Expert, the interested reader may consult [K-EXPERT 1]

If these building and use "interfaces" may be sufficient in a first step of the development process of an expert system shell such as K-Expert, it appears quickly that they are really insufficient to satisfy the first aim of the designers (e.g. : K-Expert be considered as an enjoyable and useful tool for various users classes). Indeed, the question we retain can be formulated as follows : How people getting accustomed to the use of systems relying on direct manipulation principles could accept to use with pleasure such elementary interfaces ?

So in the next chapters, we try to adapt the current K-Expert analyst's interface in order to make it compatible with the goal pursued.

3.2. M.1

3.2.1. Presentation

M.1 is a knowledge system building tool available on IBM PC, XT, AT or fully compatible PC. M.1 is produced by Technoledge,Inc. Originally written in Prolog, it is now implemented in the C programming language.

Conceptually, M.1 is an interesting cross between Emycin and Prolog. It can be interfaced to existing softwares such as data base management systems, communication networks, computer-aided design systems.

According to the M.1 reference manual [M1 87a], M.1 can be used quickly and effectively by non-specialist computer programmers with no prior experience in knowledge system technology. A single programmer can typically builds a first useful system using M.1 within two to three months. But M.1 is also targeted at programmers who wish to develop expert systems that can be easily integrated into conventional computer environments.

Knowledge systems built with M.1 are designed around a knowledge base relating to a particular task or application and an inference engine that performs the reasoning process to solve specific problems in that application area. The knowledge base can contain up to 2,500 rules, facts and meta-facts. The facts are represented as attribute-value pairs and can be followed by certainty factor (measure of likelihood between 100 (true) and -100 (false)).

The relationships between facts are defined by IF-THEN rules. The conditions of the IF part of a rule are connected with the boolean operators AND, OR, NOT . They are either of the form "EXPRESSION = VALUE (optional) cf CF (optional)"or they are meta-propositions (for example, an instruction to test whether some expression is known). The conclusion of the THEN part of a rule consists of facts connected by the boolean operator AND. Certainty factors can be associated to the rules. M.1 also accepts variable rules. They allow rules with a repetitive pattern to be collapsed into a single rule. The sets of values that could be substituted into these rules are included in a "lookup table" in the knowledge base.

Meta-facts provide informations useful in determining an expression value as the "question (expression = "what is the value of expression") " instruction. This meta-facts do not tell M.1 the value of an expression but tell it how to ask the value .

During a consultation each rule can be used by the system one time except for the variable rules. The control strategy of M.1 is the backward chaining mode. It works with input data and rules in the knowledge base to deduce facts or conclusions. The deduced facts are stored into the cache. Limited forward chaining can be accomplished. Indeed, when conclusions match a specified pattern, a special set of high-priority goals can be activated. For example, by using the WHENFOUND instruction in a rule, the analyst has the ability to write a kind of single condition rule which will trigger off an action such as pursuing a new goal anytime it becomes true. Having done this, M.1 would then resume the backchaining where it had left it off.

To conclude this general presentation, we can add some remarks about the interface. The screen is always divided into three major area : an action bar and pull down menus, a panel body and a function keys area.

The action bar appears at the top of the screen, is permanent and contains a list of choices. When users select one of the choices, a pull down is displayed that lists available actions.

The panel body is located below the action bar and it can be divided into several panel body areas if the application needs to show users more than one group of information at a time.

The function keys area appears at the bottom of the panel. It contains a list of function keys assignments. The interaction with the screen are performed only with the keyboard.

3.2.2. The Knowledge engineering interface

M.1 provides two interaction environments : one for the knowledge engineer who is developing an expert system (e.g.: the development environment) and one for an "end-user" who consults an instantiated expert system in order to gain expert advice(e.g.: the delivery environment).

When M.1 is launched , the delivery environment is displayed on the screen. The design of this screen is presented in Figure B.1 (Appendix B).

By pressing <F9>, the analyst is allowed to switch back and forth between M.1 delivery and development environments. The interested reader may find a figure illustrating the development environment in the Figure B.2 (Appendix B).

The user may issue M1 commands via the pull-down menus which contain the whole available functionalities.

3.2.3. Summary of the main available functionalities.

Let's remark that this list is not exhaustive.

3.2.3.1. Functionalities related to the building of the knowledge base.

Any text editor that can generate ASCII text can be used to build the knowledge base. This gives the analyst the advantage of using a familiar and powerful environment to describe the attributes and to write the rules.

But the Knowledge base can also be modified within M.1 development environment. By this way, a new entry (it means either a fact or a rule or a meta-fact) can be added at the beginning or at the end of the loaded knowledge base. Likewise, an entry can be removed or modified.

Other actions can be performed on the current knowledge base such as either to clear it and reload another one or to load a knowledge base at the beginning or at the end of the current existing one.

At the end of all these modifications, the analyst can ask to save the current knowledge base into a file. This one is in text format, so it may be inspected and edited using any standard text editor.

But another opportunity is provided to the analyst. This last can ask to save only the new or the modified entries. This option enables him to reintegrate these entries into the master knowledge base at a desired place by using a text editor.

Of course, in order to facilitate the analyst's task, M.1 provides a set of lists such as :

- the list of all the entries of the knowledge base,
- the list of all the entries that can be used to find a value for a specified expression,
- the list of all the entries that can be used to conclude a specified value for a specified expression,
- the list of all the rules in which the value of a specified expression is used in the premise to find the value of some other expressions.

3.2.3.2. Functionalities related to the consultation of a knowledge base.

The analyst may start the inference engine in three different ways.

Indeed, He can ask to start a consultation after having cleared the cache. This last is the repository of all the conclusions made in a previous consultation.

Another opportunity is to start the consultation without affecting the contents of the cache. This is useful to restart a consultation that has been aborted or to start up an end-user system that requires a pre-loaded cache file.

The last option is to clear the cache of all the entries concerning a specified expression and to start the inference engine to find the value of the given expression.

Obviously, the user can abort a consultation at any time without affecting the knowledge base and the cache. He can also exit the consultation and return immediately to the Dos system prompt.

Moreover, during a consultation, the analyst can receive an explanation about the expression which is being sought and a list of the legal values which can be given to this expression.

3.2.3.3. Functionalities related to the debugging and the trace

During a consultation, M.1 has an inference tracer. When the option "trace" is on, M.1 produces and displays a text "trace" which is a report detailing the inference engine process and the events of the consultation. This information includes the tried rules ,the rules that have succeeded or failed and the conclusions that have been noted in the cache.

The user can also ask to activate the trace only when M.1 finds a specified value for an expression. Obviously, the analyst can desactivate this trace or all the tracing whenever He wants. Punctually, He can always ask M.1 to show which expressions are currently being traced.

The trace can be saved in a specified file and then, it permits to save the behavior of the knowledge system. But it can also be read by external programs and it provides a mechanism to integrate M.1 with other softwares. Moreover, the trace can be printed.

To support the debugging of a knowledge base, there are other options. The user can ask (by setting the panel option on) to display five areas which show the activity of the system. Those areas display the events as they take place during a consultation, the conclusions that are noted in the cache, the knowledge base entries as they are noted in the cache, the knowledge base entries as they are tried during a consultation, the acceptable responses to questions that are generated and the dialogue.

Moreover, the user has the ability to regulate the speed of the displaying of the informations in the described window. This one is illustrated by Figure B.3 (Appendix B).

3.2.3.4. Functionalities related to the cache.

We have already defined the cache as the repository of all the conclusions made in a consultation.

The analyst has always the ability to move the contents of this cache into a specified file or conversely, to load the contents of the cache from a given file. But, He can also alter the contents of the cache. Indeed, He has the ability to add or to clear any conclusion which corresponds to a specified

expression (and a specified value and a specified certainty factor). The analyst can also clear all the entries recorded in the cache.

But other opportunities are offered to him, such as the ability to show the contents of the cache or to display the values, if there are any, of a specified expression.

3.2.3.5. Functionalities related to analyst information features.

The analyst can ask M.1 to display how many facts and rules are in the cache and in the knowledge base and how much memory is free.

A help function is available to the designer. Indeed, M.1 can display a list of available commands or a brief explanation of a specified command.

3.2.3.6. Functionalities related to the creation of a user interface.

Facilities exist for tailoring both the Knowledge base and the interface to suit a particular application and user. The knowledge base may contain enhanced or customized explanations by using the already mentioned meta-facts (e.g. the "question" instruction which enables the analyst to display an appropriate question). In addition, with meta-facts, it is also possible to disable M.1 commands in the delivery environment to protect the integrity of the knowledge base. It is also possible to create a configuration file. This is done by running the "configuration manager" which is a stand alone program. This one allows the knowledge engineer to create own menus, to add, to remove or to rename commands, to modify prompt in order to make the system more friendly to particular users.

Moreover, as M.1 allows knowledge systems to be readily integrated with existing systems, M.1 can invoke subroutines written in assembly language or in higher-level languages such as C so, a programmer can try to design a more friendly interface. For example, in order to display graphic, an external procedure has to be written in C to create the display and also to remove it before continuing with the consultation.

3.2.3.7. Functionalities related to adaptation of the development interface to the needs of a particular analyst.

The analyst can tailor his environment by using the already mentioned "configuration manager".

3.2.4. Basic interaction ways

To interact with M.1 means to work with a single window which covers the whole screen. The standard form of this window has already been presented (an action bar and pull down menus, a panel body and a function key area). The right side of the function keys area is dedicated to the display of the status of M.1 (ready,loading,consultation...). During the consultation of an expert system, the knowledge engineer will meet three different screens :

- the delivery environment or end-user interface. (Figure B.1 in Appendix B.)

- the development environment in panels mode. This means that the panel body is divided into five area in order to display the dynamic trace during a consultation. This window has already been described in the debugging features. (Figure B.3 in Appendix B.)

- the development environment in standard mode. In this case the panel body is not splited. (Figure B.2 in Appendix B.)

But the contents of the action bar, of the associated menus and of the function keys area are the same for each of these three screens.

There are two available interaction styles in the M.1 environment : the menu selection and the command language.

Menu selection.

The pull-down menus which are permanently available provide an easy accessible listing of all the M.1 functionalities. They are organized to offer groups of functionalities semantically related. The menu items are in alphabetic order. Maybe, it would be better to display them in the order of the most used functionalities.

The keyboard is the only way to interact with the menu. The designer has three ways to trigger off a function. First , He has to press on the <F10> function key to be in the menu mode. Then by using the cursor keys, He can move back and forth among the menu. When a menu item is highlighted, He can "select" it by pressing on <return>.

Another way of moving around within menu is to use the fist letter of the desired menu entry. When there is more than one entry starting with the same letter, typing that letter again will move the cursor to the next entry with that same first letter.

However M.1 provides also accelerator keys that enable the user to quickly perform some of M.1 frequently used commands.

All the items of the menus lead to an action but for some of them, M.1 needs to obtain additional informations. In this case, a temporary pop-up box appears on the screen to accept the user's input. The bottom of the box contains a function keys area.

M.1 disposes of four forms of pop-up boxes :

- File box which lists the files in the current directory. A highlighted bar appears over the first entry and the use of cursor keys or the typing of the first letter of a file positions the bar on the desired file name. The analyst can also decide to load a file in another directory. In this case, He has just to type the Dos path.

- Confirmation box. This one appears in order to ask the analyst to confirm a command He has just issued. The analyst answers by typing an "Y" or an "N".

- Prompt box which appears if M.1 needs additional informations in order to complete a command.

- Edit box. This box allows a user to add a new knowledge base entry or to edit an existing one. The user can type multiple lines of text inside it.

Command language.

This one is only available in the development environment. It enables the analyst to issue a command by typing it directly in front of the prompt. These commands trigger off the same functionalities as the menu items. The general syntax of the command is the following one. The verb comes first and the object of interest follows. The verbs of the commands are the same than the verbs displayed in the menu items.

3.2.5. Initiative and control

The analyst is the initiator of the actions rather than the respondent. Indeed the analyst usualy has the feeling that He is in charge of the system and that the system responds to his actions. When M.1 is started, the analyst can undertake one of the commands displayed in the menu items and He organize his Session as He wants. The system does not impose any pre-defined sequences of actions. The analyst has to answer system questions only if a pop-up box is displayed. But this one appears if some additional informations are needed to perform an action initiated by the analyst himself.

However, we do not think that an analyst has a full control on his work. Indeed, the user cannot do everything He wishes and cannot visualize his task. The biggest problem is that the functionalities offered in order to help an analyst to alter the knowledge base are not adequate. An analyst can only either add a new entry at the beginning or at the end of knowledge base or replace an existing one. So, in the M.1 environment He is not able to choose the place of his new entry. If He wants to do so, He must save all the new entries in a text file and quit the M.1 environment. Then, with a performant text editor, He can integrate the new entries where He wants into the knowledge base file.

Another example illustrating the fact that the analyst has not a full control on the system is the following one. In the M.1 environment, the analyst can list the contents of the knowledge base. In this case, the whole contents is displayed in a scrolling way at the screen but the user is not able to stop it whenever He wants. He has to wait that the end of the file display be reached. Then, if He sets the system in scrolling mode, He can try to use the arrow keys to return to a particular entry. But, the available part of the file which is accessible depends on the scroll buffer size. The user can change the size of

such a buffer to enable the system to scroll the contents of the whole knowledge base but once again, the analyst has to quit the M.1 environment and to start the already mentioned configuration manager. We can add that the user is not allowed to modify the knowledge base in the displayed list.

As the M.1 environment disposes of only one window, the analyst is not able to interact at the same time with the listing of the knowledge base and the edit box (which allows him to add or to modify a knowledge base entry). So if an analyst wants to create a knowledge base or if He has fundamental changes to do on it, it is better for him to leave the M.1 environment and to work with an external editor. This is not a very attractive solution because each time He wants to test his knowledge base, the analyst has to enter the M.1 environment again.

A lack of control is also due to a lack of additional functionalities. Indeed an analyst has the opportunity to create some file (for example when He is running a consultation, the trace can be written in a file) but He can never delete it inside the M.1 environment. Sometimes, as for the previously given example, He is not able to read it again in the environment.

3.2.6. Flexibility

3.2.6.1. Internal flexibility

By internal flexibility, we mean the opportunity to access the same functionality by more than one way. In our particular case, all the available actions are accessible through menu items and are on the same level. There is only one way to undertake them and this one has already been explained in the interaction style paragraph. This internal flexibility seems to be sufficient.

3.2.6.2. User flexibility

In this section, we determine if M.1 is appropriate to the novice, intermittent and frequent users.

M.1 offers a menu selection interaction style. This one is appropriate for the novice and intermittent users but can be appealing for the frequent users. Indeed the action bar and pull down menus provide the advantage that all the actions available to users are visible and can be

requested by simple interaction techniques. It helps users to find the desired action without having to remember the name of the action.

For the frequent users, M.1 provides accelerator keys and also a command language.

So M.1 seems to be appropriate for the three classes of users. Moreover the environment enables an analyst to progress by using it. Indeed, the accelerator key are displayed in front of the corresponding functionality in the menu-item and the command language try to be the more closer to the menu-item syntax.

But we think that this tool could be more appropriate to the novices if masks were displayed to the analyst whenever the latter wants to add or to modify an entry. Indeed, M.1 offers only a free text editor which does not give informations about the needed syntax of an entry.

3.2.7. Feedback.

For every analyst action, there should be the same system of feedback. M.1 does not respect this rule for every action. For example, when an analyst wants to log the trace to a file, a message tells him that the system is creating a file. But when the analyst decides to log the trace to a printer, no message is displayed.

If the analyst wants to trace an expression, the feedback will only occur during a consultation when the system meets the given expression. As for the "trace on" and "trace off" commands, out of a consultation, there is no feedback and it is impossible to determine if the trace is set or not. The user must start a consultation to know the trace state. A "bistable" menu item should solve such a problem.

However when an action needs a quite long time to be performed by the system, a message such as "loading" or "reseting" is displayed in the area which displays the current status of M.1 (right side of the function keys area).

3.2.8. Errors handling.

When an error occurs, a message is displayed on the screen in front of the prompt. This one explains what is the error but it also displays a message number which has no meaning for the user. Indeed there is no table with the corresponding error message number in the M.1 reference manual. For example, we have received the following message : " ERROR 105. Possible M.1 internal error. Please report to Framentec product support (155) ". In such a case, the only thing the analyst can do is to restart M.1. This fail of readability can frustrate the analyst.

Moreover, the performed errors are often syntactical ones which occur when a user types a command. But when an error occurs the system does not display the previously introduced command. So,for the correction, the analyst has to type it again.

3.2.9. On line help

M.1 provides an on line help but only to list all the available commands and to provide access to a brief explanation of them. However, no help is provided to the analyst about the syntax of the entries in the knowledge base.

3.2.10. Memory load.

In order to initiate a command, the memory is not overloaded because all the available functionalities are displayed in the menu items as well as their corresponding accelerator keys. Moreover, the command language is not far from the menu items syntax and an help is available to list all the commands. But the memory is overloaded when an analyst wants to add or to modify an entry of the knowledge base. Indeed, no mask is displayed and no list is available to display the possible values of the different elements of the knowledge base entries. Moreover, as the user can initiate only one action at a time, it is not possible to edit a rule and at the same time to list the already existing rules. So the analyst has to keep in mind the contents of his knowledge base.

3.2.11. Dialogue interruption possibilities.

At any time the analyst can cancel the action He is performing.

3.2.12. Consistency.

First of all, we can mention that the whole interface tries to follow the principle of the Common User Access of IBM [IBM 82] Indeed the three parts of the window (menu, panel and function key) are those proposed by this standard. This interface is consistent with all the tools which obey to this one

Nonetheless, some consistency problems are present:

- The use of the same function key to perform different functionalities. For example, the function key <F10> has two different meanings in the proposed interface. indeed, when a pop-up is displayed, this key is used to accept it. In another case, this one is typed to switch to the menu mode.

- The Common User Access recommends to put "..." after a menu item which does not trigger off directly the associated command but needs a dialogue box in order to type some more informations. This rule is not respected in the interface.

- All the actions which could be disastrous should be preceded by a confirmation box. However, for such functionalities (like the "reset knowledge base" which clears the whole contents of the knowledge base, for example), the M.1 environment displays no confirmation box.

- The syntax of the commands or menu items is not always the same. In general, the first unit is a verb such as in "log printer". Nonetheless, it can sometimes be a substantive such as in "panel on" or "options". Nevertheless, the syntax of the menu item is consistent with the command language.

- The same functionality "quit" is accessible by two commands named "quit" and "exit". However, only "exit to dos" is present in the menu items. This is not a problem except that the chosen accelerator key is Alt-Q (as "quit").

3.2.13. Some remarks about the inputs and the outputs.

3.2.13.1. Input.

The use of default values is not implemented. For example, in a confirmation box, the user must type either 'y' or 'n'.

The inputs follow the user's rhythm (not the computer one).

During a consultation, when a question is asked to the analyst and all the legal values for the answer are known, the analyst has the opportunity to type only the first letter of the desired value instead of all of them.

The input procedure is always the same : either to type a command, to select a menu item, to answer a dialogue box or to respond a question during a consultation. As no mask is displayed, the analyst can never choose to fill a field before another one.

3.2.13.2. Output.

The response time seems to be acceptable but we have tried to use M.1 only on a small knowledge base.

The display rate is sometimes too quick to enable the analyst to read. Indeed, when a not interruptible scrolling is displayed at the screen, the speed is too fast.

If the screen of the computer accepts to display colors applications, M.1 can also use colors. But the chosen colors are not the best ones to make the reading easier.

3.3. NEXPERT (VERSION 1.1)

3.3.1. Presentation

Nexpert, a rule-based tool proposed by Neuron Data, is primarily targeted at the research community and offers a highly graphic development interface making full-use of multi-windrowing advantages. This tool is defined as a hybrid system in this sense that it supports both a reasoning system and a powerful object-oriented representation.

This software is available on 512K Macintosh and IBM PC AT. It has been developed originally in Lisp but has been converted into a mixture of C, Pascal and assembly language later.

The inference strategy of Nexpert is really powerful ; it allows the analyst to write rules without specifying forward or backward chaining. The rule format is the following one :

IF ($condition_1$ AND ... AND $condition_K$)

THEN (hypothesis or goal which becomes true when conditions are met)

AND DO ($action_1$ AND ... AND $action_L$)

In particular, these actions can concern the change of the value of one or several data ; the creation and deletion of objects and links ; the reading and writing in Data Bases ; the display of graphics and text ; the affectation of the inference engine ; the reset of values ; the execution of external programs and the loading of new rules.

The expert systems developed with Nexpert can be consulted in 3 ways which are forward consultation, backward consultation and mixed consultation.

The objects are elementary units of description composed by properties and grouped in classes of objects sharing properties. This notion introduces the concept of hierarchical representation of objects with inheritance opportunities in Nexpert.

Nexpert provides a formalism to express rules and objects which are saved in files called knowledge bases. We want to note that facts which characterize the actual case being processed by an application are saved in external files or Data Bases.

So, Nexpert offers Data Base links in order to allow an application to retrieve external data or to write reasoning results outside. This is interesting for many reasons. The first one is that the size of knowledge bases stays reasonable. The second one is that facts data bases may be very large. The third chief reason is that facts can be accessed or produced by other applications.

The Nexpert architecture is event-driven ; it does mean that it can integrate messages from the outside world or external programs, which themselves might have been triggered off by Nexpert rules or objects.

According to the reference manual of Nexpert about the interface philosophy, we want to emphasize that "information provided to the analyst by the intelligent system must be displayed in the most expressive way possible, making full use of human visual and perceptual capabilities" [NEXPERT 87].

This interface allows a rapid incremental development with direct access to the reasoning mechanism. Moreover, "it establishes an essential cognitive continuum between the tool with its representation, and the analyst with his or her mental model of the task" [NEXPERT 87]. Finally, we can say that Nexpert revealing at any time the different facets of the application in process, these windows are designed in order to enable users and analysts to input values in the system and to provide various levels of information.

3.3.2. The knowledge engineering interface

When one starts to work with Nexpert, the first screen which appears is that presented on Figure C.1 in Appendix C.

The so-called Nexpert window stays constantly on the screen, is permanently accessible and enables the analyst to access basic classes of functionalities. Indeed, whenever He clicks on one of the six icons a corresponding menu of options is displayed and the analyst can select a desired option by using the mouse.

3.3.3. Summary of the main available functionalities

This Section presents a not exhaustive list of the primary functionalities of Nexpert. We do not go deep into details for what concerns the objects manipulation because it would bring us too far from K-Expert.

3.3.3.1. Functionalities related to the building of knowledges bases

First, it is naturally possible to load a knowledge base from a list of the available ones and to save a knowledge base under a specified name. For these two options, a directory path can be specified.

Of course, an analyst can also select the current knowledge base among the loaded ones, clear the specified knowledge base from system's memory and change the knowledge base an object belongs to.

A Rule Editor (illustrated by Figure C.2 of Appendix C) enables the analyst to create, modify and delete rule from a mask. He can also copy hypothesis, data, class, object, property and function selected in a corresponding list to the edited rule component . However, the copy of a whole Rule is not implemented.The analyst is always able to cancel the modifications he has performed on an edited rule. A syntactical control of the edited rule is realized when it is saved or any time the analyst wants it.

To facilitate the creation and the modification of an edited rule, each field of the mask provides an access to a List of possible values. .

There are also other editors offering the same functionalities adapted to particular concepts such as context editor, object editor, class editor, property editor and meta-slot editor.

Two sets of functions supply a complete set of Notebooks listing the different structures present in the current knowledge base. The first set concerns the elements directly involved in the inference plane like Rules, Hypothesis and Data. The second set of functionalities concerns the elements involved in the representation plane like Objects, Classes and Properties. It's interesting to note that each list gives a direct access to the corresponding editor functionality.

The possibilities to print to a selected printer and to write to disk files notebooks editors and networks (all or only the current page) are also available.

3.3.3.2. Functionalities related to the consultation of a knowledge base

Before starting the inference engine, the analyst may choose forward (respectively backward) consultation modes and must provide the necessary informations such as initial data (respectively the hypothesis to test) and reset all the elements concerned by a consultation to the "unknown" state,

if necessary. At any time, a consultation can be interrupted and obviously resumed.

A default setting of the inference engine and inheritance mechanisms parameters is also included in Nexpert system.

Whenever the inference engine asks informations to a user, this one may access to a graphical or textual documentation about the item for which He must input a value.

During a consultation, the analyst may access to a "why" function. It displays a window providing a basic explanatory mechanism including text file which may be defined while editing the current rule. It furnishes also the ability to see which is the current rule and to browse along the hypothesis links by using so called "why" and "how" buttons.

Moreover, a Rule Network can be accessed, at any moment of a consultation. In particular, this Network can focus on the rule currently under evaluation. Two functionalities concern the Rule Network facility. The first one opens the rule network window and gives access to the following listed abilities :

- deductive navigation in the network (the hypotheses evoked by a datum) ;
- evocative navigation in the network (the rules which lead to an hypothesis) ;
- erase elements of the network and their links to others ;
- focus on the current network on a particular structure (new group of rules) ;
- display graphic and/or text files associated with a particular item ;
- obtain a limited development of the network ;
- clear the network ;
- undo the last modification performed on the network ;

- view a line of the network.

The second functionality is called Overview Rule Network. It opens the Rule Network Overview mechanism allowing the analyst to browse rapidly through the current display of the network.

Finally, two functions similar to those related to the Object Network are also proposed. They are respectively called "Browse Object Network" and "Overview Object Network".

3.3.3.3. Functionalities related to the debugging and trace operations

The Browse and the Overview functionalities of the two considered networks are obviously accessible during performing a debugging session. Moreover, it is also possible to place breakpoints graphically in the network and to add icons to each item of a network. This last possibility makes possible the display of the network elements status to the analyst at any moment of an execution.

An Access to the different lists and editors appearing on the network is allowed at this level. This function makes possible a direct correction of the analyst's work.

Many visualization possibilities are also available. We want to note the ability to display :

- a trace of the execution ;
- the hypothesis currently under evaluation ;
- all the conclusions the system has reached up to that point in the inference ;
- the rule currently under evaluation ;
- the list of known attributes and suggested or generated hypotheses ;
- the list of known hypotheses along with the confirming conditions or the counter arguments ;

A complementary Journal function implements two different functionalities : the recording and replaying of a session and the saving and restoring of a whole current state

3.3.3.4. Functionalities related to analyst information features

An "Apropos" function displays graphic and/or text files associated with a particular item. For example, one can associate a significant text to an object in such a way that when this one intervenes, the analyst or user can consult explanations about its role, meaning ....

3.3.3.5. Functionalities related to external elements

It's possible to read a file from a Data Base using dynamic queries, and to write a file into a Data Base An execution of external programs is also permitted by the Nexpert system. Among the external files, we find spreadsheets, relational Data Bases and Data Base files.

3.3.3.6. Functionalities related to the creation of a User interface

Before a consultation begins, a Set up functionality makes possible the indication of all the windows that will be accessible for a particular user.

An almost obvious remark is that the user interface is really an elementary one. In fact, it is similar to the consultation of a knowledge base by an analyst. The principle is the following one : what is accessible to a user of a particular expert system is the "Session Control window" (see Figure C.3 of Appendix C for a visualization of it), the related functionalities (such as suggest, volunteer, use a journal, why option...) and some other information windows if the analyst has selected them by an environment setup.

When a session starts, the user has just to answer system questions displayed in the upper part of the Session Control window by selecting a value proposed in an interaction window appearing in the lower part of this window. He can always type directly a value instead of selecting one. What can be said is that this interface corresponds to the minimal implementation to provide to people having to consult an expert system. It is not difficult but also not very attractive to use. It can not be offered less but of course, it could be offered more.

The Run-time interface as it appears now can not be adapted to various users classes. Of course a "made-to-measure" user interface can be implemented by an analyst . But, to do that, this person must be a competent C programmer ! Moreover, this programming task can be very time-consuming. Only with these programming efforts it can be possible to provide the user with the ability to work visually and by the same occasion to build a direct manipulation user interface. By working visually, we mean to input values (in order to answer system questions) by graphical manipulations on meaningful visual task objects representations.

3.3.3.7. Functionalities related to the adaptation of the development interface to the needs of a particular analyst

In order to adapt his interface, an analyst can choose the settings of the Nexpert environment for a session (example : colors, size of network elements ...etc). He can also indicate which printer is going to be used. In order to obtain the same environment any time the system is accessed, it is possible to save the current configuration of the system.

3.3.4. Basic interaction ways

As it has been said in the presentation of Nexpert, it appears that this software interface is highly graphic and relies mainly on windows. There are overlapping possibilities, but only one window is active at a time. In this Section, we intend to present the primary techniques on which relies every interaction sequence.

3.3.4.1. The standard windows

The standard window is the general structure in which more precise functions are implemented.

It is useful to give some words about scrolling mechanisms which allow to visualize the hidden contents of a standard window. The classical scroll bar mechanisms (elevator and arrows) permit the user to navigate inside the contents. Some standard windows in Nexpert have also a page flipping mechanism to facilitate browsing. This mechanism is located at the left corner of the windows presenting information in a page by page fashion. Moreover, lateral alphabetic indexes are included in some windows. They make

possible a direct access to the first item corresponding to the chosen letter. Finally, the zooming mechanisms called overviews are used to rapidly move inside a very large graphic display like a network. A dotted rectangle indicates the current area covered by the window in the current mode. This dotted rectangle and consequently the current area of the window can be moved by using a mouse.

The standard windows can also contain "controls" implemented to provide basic commands graphically. In fact, these "controls" are buttons which correspond to a function and initiate this one immediately when they are clicked. These buttons are located into a sort of command line under the title or are independent and thus can be located anywhere in the window.

When the standard windows are used to input information in the system, special areas are designed in order to show the user information and to allow him to edit it. These particular areas are called "check items". They consist of the check boxes which display a choice among alternatives and graphic checks which are similar to the previous ones but whose shape is strongly related to or indicative of the function they represent.

3.3.4.2. The Dialog Windows

These windows are used to provide the Nexpert system with information. This is performed by means of settings buttons, selections of one or several items in list boxes and setting of numerical parameters

They require an user's reply (like a click on an OK or CANCEL button) and while they are opened, it is impossible to access to other windows.

Among the Dialog Windows, we can distinguish between the Message Dialog Windows and the Item-List Dialog Windows.

The Message Dialog Windows contain a message or a warning concerning a potential result of the user's current action or a request for specific information.

The Item-List Dialog Windows display a list of items to be chosen by the user for further processing.

3.3.4.3. The pop-up menus

The last type of windows in Nexpert is the "classical" pop-ups. They appear temporarily on the screen, inside a window and provide a choice of immediate actions. They are characterized by their adaptability. They bring up choices dependent upon the current state of either the whole environment or of a particular item displayed in the window. Another important characteristic is their "hysteresis" : when the mouse is moved too far away from them, they disappear.

Among pop-up items, there are "bistable" ones. Their value change when they are clicked on. For example, when one clicks on "show transcript", the transcript window is displayed and the item becomes "hide transcript".

3.3.4.4. Icons and graphic visualization

These graphic elements are used to fulfil 3 goals. The first one is to give an easily understandable representation of functionalities (instead of buttons)

Example :

The eraser pictured beside the network enables the analyst to erase a node in this network.

The second is to provide a graphic documentation for some items ;

Example :

Little significant icons can be associated to any link of a network in order to reflect its status at a step of a consultation. So, a check mark means "True" while a question mark means "Unknown" (an item which has not yet been investigated).

3.3.4.5. The mouse

The use of mouse in Nexpert is fundamental. It seems impossible to work without using it "a minimum". This necessary mouse is composed of 3 buttons, each one has a particular function :

- The LEFT button is used to select an item ;

- The MIDDLE button brings up a pop-up called "windows". This is an important ability as we are going to see it in Section 3.3.5.

- The RIGHT button brings up a pop-up corresponding to a particular item when the mouse pointer is located on it and to a particular window when the pointer is not on a particular area of a window.

3.3.5. Initiative and control

Generally speaking, we can say that the analyst is free to undertake everything He wants by using Nexpert. Of course, this freedom is restricted to the offered functionalities but nonetheless, these ones are presented in such a way that He can organize his work as He wishes. Indeed, at any time the analyst can have access to the six big classes of functionalities of Nexpert from anywhere just by activating the Nexpert window and clicking on the adequate icon, by activating an already opened window or by using the middle button of the mouse. There is no interdiction. Except the display of the so-called "Nexpert window" at the beginning of each work session, there is no predefined using scenario.

In a word, we can say that the analyst initiates actions much more than He replies to computer system questions. This opinion is enhanced by the idea that Nexpert tends to direct manipulation. In effect, there are many examples of recourse to visual representations of task or computer concepts thorough Nexpert interface.For example, thanks to the masks, the analyst, using a mouse, can think that He acts directly on the different components (and subcomponents) of a rule.

From the manipulation experience we have had the opportunity to acquire by "playing" with Nexpert, it appears that the learning process can be quick because we work on and with visual significant elements and because we can perceive immediately the effect of any action. For example, a rule modified via the mask editor is immediately updated as well as the related already activated windows like the rule network.

This learning is also made easier because the icons in the network are similar to those appearing in well-known programs like MacDraw. Besides this, seeing that each potentially destructive action is preceded by a warning containing a concealing ability, the analyst "feels at ease" and so He is not afraid to explore Nexpert more and more.

Nexpert has given us a convincing example of one of the many advantages of direct manipulation. This software enables us to work directly at a high level without any syntactical learning and only with a small mental decomposition effort. This leads us to suppose that the retention of Nexpert in time should be satisfactory.

From analyst's experience, it appears that the implementation of a multi-windows system is particularly pertinent in this sense that it reflects his mental proceeding. This way of doing seems to offer all the advantages underlined in Section 1.3..

From practise, it is clear that the windows management is not always easy. The analyst has to resize and to move windows in order to have simultaneously access to the information which is contained in each one. There is the inevitable overlapping problem. However, this last one is partially solved thanks to the ability to see and to access to all the activated windows at any time and from anywhere by using the windows pop-up.

It seems that a real control feeling can emerge when an analyst works with Nexpert. We are just going to illustrate this point by some striking elements :

- At any time, the windows pop-up is accessible just by clicking on the middle button of the mouse. By looking at the contents of this pop-up, the analyst can directly see all the windows already opened, He can activate them or access to functionalities of the permanent "Nexpert window".

- As this "Nexpert window" is permanent, it is impossible to close it inadvertently and so to loose every control on the developing process led with Nexpert.

- The feedback procedures being not neglected as it is shown in Section 3.3.9., the analyst is not confronted to anguishing questions concerning what is happening ?

- There are no static limits imposed to the analyst. Indeed, in the rule editor, for example, there are no limits to the number of actions that can be input for a single rule. Obviously, there is a limited number of actions

displayed at a time. The same is true for the contents of each field of the mask. In the mask, only a limited part of the field value can be visualized, but one can always access the whole value by making the desired field contents appear in the edit line which is located in front of the mask.

- Pop-ups are organized in such a way that they offer groups of functionalities semantically related. So, the analyst never gets lost and He has not to look for an option in many places. Moreover, pop-ups propose such options that the analyst can access directly to a function interesting for the accomplishment of his current task without having to switch to the general Nexpert window.

We conclude this point by some thoughts about the transparence of the computer. In fact, if we think to what has been explained in the theoritical part of our dissertation, it seems that the best interfaces should go hand in hand with the perception of the computer as a tool, which disappears under the actions performed on task concepts.

Here in Nexpert we are not sure that this transparence is completely reached because some purely technical concepts must be manipulated by the user such as the writing of print commands for the printer, the switching between directories and subdirectories.

Now, the remaining question is the following one : Should these elements disappear ? The answer is not obvious because it appears that these two technical elements offer powerful abilities to the analyst and moreover, with the diffusion of the computer science in many areas, more and more people learns to manage efficiently notions such as the concept of directories !

3.3.6. Flexibility

3.3.6.1. Internal flexibility

As it has just been said, there are many ways to access the same functionality and the same functionality can be reached from many places. As this ability has already been evoked previously in Section 3.3.5., we do not detail it again.

3.3.6.2. User flexibility

Is Nexpert adaptable to many users classes ? Is it possible to make it evolve according to the learning process of the users ? These questions seem more delicate to answer.

Indeed, according to our readings, it appears that seeing the existence of three typical users groups (e.g. novice, expert and intermittent users), a good interface should offer many using mode such as command languages, abbreviations...etc.

When we look at Nexpert software and manual, and when we speak with analysts working with this expert system shell, we are confronted to an absence of these things. Apparently, there are some abbreviations but they are not easy to find and not clearly explained in the manual. Among the abbreviations, we find the classical but useful ones such as the ability to replace a click on an default OK button by a press on the Return key. Moreover, there is no command language.

Of course, direct manipulation characteristic makes Nexpert interesting for all users profiles but sometimes Nexpert is heavy to manipulate. The novice makes mistakes in the mouse manipulation. Luckily, these wrong manipulations are not grave at all but they are annoying. Moreover, the existence of multiple paths can be disturbing. The expert is sometimes fed up because He must always use a mouse instead of typing directly a rules file name in a normal text editor for example. However, practice shows that these remarks do not constitute a real handicap.

We are going to conclude this point by signaling that each analyst is able to adapt Nexpert to his particular needs by "playing" with the settings functionalities as it has been shown in Section 3.3.3.7.

3.3.7. Feedback

The necessity of a feedback is a real concern for the designers of Nexpert. We take this affirmation from the fact that actions are not initiated by users and followed by a silence. By silence, we mean that the computer works but does not signal anything to the analyst. In fact, any period during which

the computer is working, is signaled to the analyst by the display of a message box containing an icon representing a thinking head and a warning message.

We find also feedback in this sense that when one clicks on a page corner to turn the page, one sees that one turning. Moreover, whenever a user executes a functionality affecting the contents of a not activated window, this one is updated automatically. The same principle is applied to bistable pop-up items whose value is updated when an event which affects them occurs.

The function buttons follow also the same philosophy. A button is dimmed up to the moment it can be used. As an example,one has just to think to the OK button of the rule editor. This one is only accessible when the rule is completed.

We can also mention the fact that when the user selects a particular option in the network for example, the cursor changes its shape according to the chosen option.

Finally, let's remark that there is a problem when one tries to resize windows. Indeed, if there is a rank of buttons under the title, some can be hidden at the completion of the resizing. What is problematic is that nothing mentions their presence. Moreover, it is no more possible to click on them.

3.3.8. Errors handling

From what we could observe, an errors handling system is implemented. This one is particularly interesting in this sense that in case of problems, clear and directly understandable sentences are displayed. They explain the situation and suggest a solution. There are no esoteric references to errors numbers. From experience, it appears that the ability to use a mouse to select appropriate values in a list (displayed in response to a click on the right button of the mouse when the pointer is set on the considered area) reduces the amount of mistakes (typing errors...etc).

3.3.9. On line help

An online help ability is not furnished. This can be a serious lack for novices or intermittent users. In case of problem, when the user is not completely sure about how to do something or about the use of a function, He must look into reference manual or start a trials-errors process. This last one

can be used because of the protection from dangerous functions by warning messages but of course, it is not the most efficient method.

3.3.10. Memory load

The long term memory is not overloaded in the sense that because of its direct manipulation foundations of Nexpert, the analyst can visualize his work.. The icons, the buttons and the pop-ups are helpful in order to reduce the memorization effort to a minimum.

The short term memory is also not overloaded notably because of the lists, the networks and overviews, the clear presentation of input and output values in meaningful windows. As the analyst can access to everything from anywhere, He has not to try to keep in mind many informations. Because of this, He can concentrate himself on the task accomplishment.

3.3.11. Dialog interruption possibilities

- Canceling : is implemented for each modification operation ;
- Deleting : is implemented for each element of an expert system (from the knowledge base itself to the rules, objects...etc) ;
- Start again : it is possible to restart a consultation ;
- Finishing : at any time, it is possible to leave Nexpert and to return to the Dos (by clicking on the "Quit" option of the "system icon" in the Nexpert window and by answering to a warning message).

3.3.12. Consistency

Except of small points, we can consider that the consistency of the interface of Nexpert exists. Among these problems ,we note :

- The use of the same term to give access to different functionalities.

Example :

The function "Cancel" in the editors has an effect similar to this of the "Undo" function in the networks.

- The same functionality is designated by different terms.

Example :

The function "Quit" in the system icon brings back the user to the Dos but the function "Quit" in the editors closes the corresponding window and brings back the user to the last activated one.

- The rules list displays a rule per page but in the other lists, the pages contain all the items beginning by the same letter.

- The little square which may appear besides the items of a list has not a uniform meaning. Indeed, a square besides a file name in a list indicates the file that is going to be saved. While a square appearing beside a property in the properties editor gives access to the corresponding meta-slot via the meta-slot editor.

- The three points following some items in pop-ups are not apparently used uniformly. Indeed, at a first look, we could say that they are used in order to signal that the item gives access to something that looks like a formulary while items without three points have a direct effect. But, if that is the meaning assigned to the three points, why do not we find them after items such as "Set up environment" which opens a dialogue box when it is clicked ?

- The action syntax is also not uniform everywhere. In some windows, one has to select first an object by clicking on it and then one can accomplish a related action. For example, to clear a knowledge base, one must select one in a list and after this, one has to click on the "Clear" button. In other windows, like the Rule Editor, one selects an action first ("Modify" ,for example) and then one can designate the field to modify by clicking on it.

3.3.13. Some remarks about inputs and outputs

3.3.13.1. Input

- The use of default values is implemented. For example, the printing command has just to be typed when the "Print" option is selected for the first time. After this initial typing, the command is saved and restored automatically when needed.

- The inputs follow the analyst's rhythm (not the computer one).

- In the mask editor, there is an edit line, this is interesting because it permits not to modify directly a field in the mask and to type fields values longer than the size of the displayed mask fields.

- The used terminology seems adequate in this sense that the task terms are coherent with the analyst's world. There are not too many computer terms ; anyway, those which are used are classic (seeing the computer science development in everyday life). The terminology problems are mentioned in the consistency point which evokes the consistency.

- The input procedures are not constraining : the analyst can fill the input field according to the desired order. There is a default order (e.g. when the analyst types on the Return key, the cursor moves automatically to the next field). Nonetheless, this order can be changed just by using the mouse to click on the desired field.

3.3.13.2. Output

- In the examples to which we have had access, the response time was acceptable. For example, the rule network was nearly instantaneously built. Nevertheless, we do not know if this is always the case with bigger expert systems.

- Among the codes used to display outputs, we find notably alphanumerical characters, symbols and "colors". The characters are uniformed, easy to read, not too tiring. The symbols (for example, the icons expressing the status of each network node) are significant, easy to understand, to read and to keep in mind. The colors are not implemented on the version of Nexpert we have consulted except the highlighting abilities. These last ones are used in a familiar and adequate way.

3.4. CONCLUSION.

Having studied M1 and Nexpert interfaces, we can emphasize on new functionalities that could be useful for K-Expert. A detailed presentation of the retained functionalities is presented in the following chapter.

We also retain some interesting man-machine interaction methods. Indeed, we think particularly to Nexpert because thanks to its direct manipulation concept, this software provides users with a set of attractive methods such as a flipping mechanism, index and overview abilities.

In the same perspective, we have decided to have recourse to a multi-window interface. Indeed, from our personal experience, it appears that such an environment makes the difference between the use of M1 and of Nexpert.

The chosen interaction methods inspired by this chapter are specified in the fourth chapter.

CHAPTER 4 :

SPECIFICATION OF THE K-EXPERT INTERFACE

Starting from the literature we have consulted about interfaces (Chapter 1), from the general study of the analyst's profile and task (Chapter 2) and also from interesting characteristics of existing expert system shells interfaces (Chapter 3), we are now going to specify our application (e.g. The building environment of K-Expert).

As this one corresponds to an interactive application and as our aim is to design an ergonomic interface, we have adopted an original method. Indeed, during the whole specification process, we have kept in mind and integrated an ergonomic dimension. The suggested process refers to some models linked to functional analysis of classical applications. In particular, we consider the model of the structuration of data and the model of the static of processes [Bod 89]. Moreover, we adapt freely a method of specification of interactive applications which is the subject of researches at the "Institut d'Informatique" at Namur. The interested reader can consult informations about this method in [War 88b].

In a first Section 4.1, we give a general presentation of our process of specification of a particular interactive application which corresponds to the building and to the consultation of an expert system. The following Sections 4.2. to 4.13. consist of a detailed description of each of the specification steps applied directly to our particular application. Finally, we propose a criticism of our proposal in section 4.14. .

4.1. SPECIFICATION PROCESS OF THE ANALYST'S INTERFACE OF THE K-EXPERT EXPERT SYSTEM SHELL

In our particular case, a classic specification does not fit correctly our needs. Indeed, we would like to define our application as an appropriate toolbox of functionalities enabling the analyst to perform his job.

For this purpose, we need to take into account ergonomic principles as well as to define the required functionalities and to precise the way to present them to the analyst. Classic specification processes do not integrate this aspect. Moreover, they are not appropriate to our case. As a matter of fact, we do not have to take care of the model of the dynamic of processes because our interactive application does not possess a global dynamic. Indeed the offered functionalities are not linked by chaining conditions seeing that all of them are considered as being at the same level.

So, let's start with a modelization of our own specification method. This one is illustrated on Figure 4.0. presented on the next page.

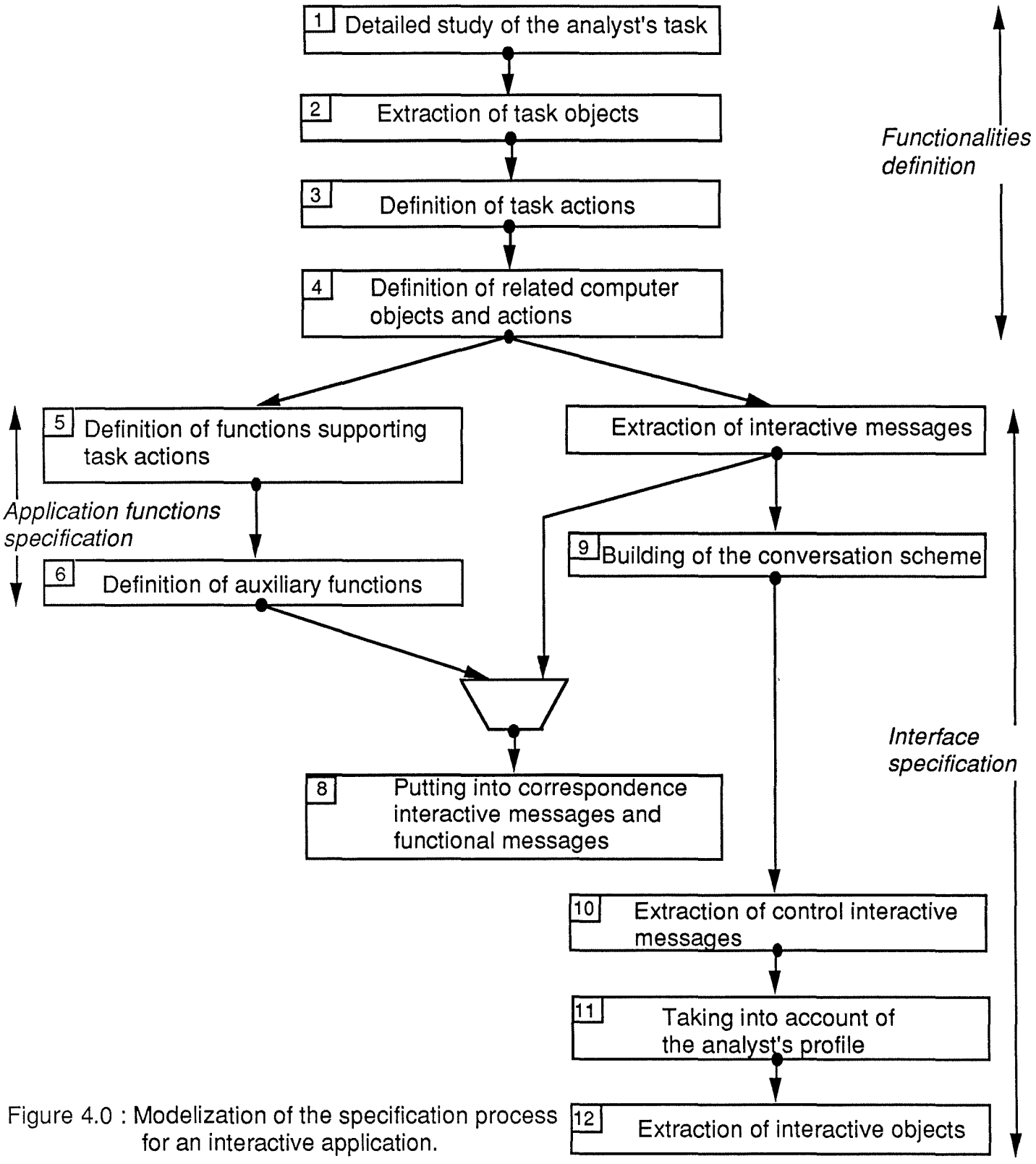


Figure 4.0 : Modelization of the specification process for an interactive application.

In a first time, as a detailed presentation of each component of the scheme is given in the following sections, we introduce only a general presentation of the envisaged method.

The global aims of our specification process are the following ones :

- Definition of the functionalities of the interactive application.
- Specification of the application functions.
- Specification of the interface

Let's say more about these 3 complementary aims.

Definition of the functionalities of the interactive application :

First, we want to define all the functionalities that an interactive application and consequently its interface, have to offer. We have already underlined that the application must be seen as a toolbox of services provided to an analyst. Consequently, this first step tries to cope with the ergonomic problem of the overfunctionality and underfunctionality of an application [Shnei 87].

As a result, in order to realize the evoked purpose, the first step of our method consists of the study of the analyst's task. From this analysis, during a second step, we extract all the objects manipulated by an analyst when He is performing his task. We call these objects task objects by reference to the Syntactic/Semantic model [Shnei 87]. When all the task objects to take into account have been described, the third stage we envisage enables us to define all the possible and necessary actions that can be accomplished on them. This step provides us with the functionalities (e.g. : actions) related to the task which have to be supported by the application and by its interface.

Nevertheless, in order to define all the necessary functionalities, we must also take into account the environment in which they are performed. Indeed, the analyst has to work with a computer. Consequently, some additional functionalities related to computer objects must also be highlighted.[Shnei 87, Syntactic/Semantic model].

After the extraction of the task actions related to the task and of those related to the environment, the two other aims of our specification process may be considered.

Let's remark that these two aims have no chronological links. They are parallel processes. Indeed, in our visualization of an interactive application, the application itself has to be separated from its interface. By this way, the specification of the application and of the interface can be supported by different kinds of people. Indeed, by difference to the application specification, the interface specification requires also to take into account some typically ergonomic concepts.

So now, in a first time,in this description of the proposed method, we explain the "specification of application functions" aim and then, the "specification of the interface" aim.

Specification of application functions :

Starting from the highlighted task actions, the 5th step identified consists of the derivation of all the application functions required to support the actions. At the same time, all the input and output required by these functions must be defined. These input-output are named functional messages.

The next step corresponds to the deduction of auxiliary functions and of their own functional messages. Indeed, the application may request some functions which are not directly linked to the task actions. A detailed explanation of this point is given in Section 4.6.

Specification of the interface :

During the 7th step of our process, we start from the task actions and objects to derive the dialogue units that must be presented to an analyst in order to enable him to perform these actions. These dialogue units are the interactive messages. They have to be linked together and their chaining has to be explained in order to express the whole dialogue. This is realized by the building of the conversation scheme which is the 9th step of our process.

This chaining constraints must also be translated into particular interactive messages which are the control interactive messages. These ones

are dialogue units enabling an analyst to decide of the direction that has to be imposed on the dialogue.

At the end of the extraction of all the interactive messages, one of the most important steps to consider in order to achieve the evoked aim concerns their visualization at the screen. This step is based on a lot of ergonomic choices deduced from the analyst's profile. This profile and the justification of the interface choices are exposed during the 11th step.

Then, in the 12th step, the interactive messages are translated into interactive objects which are in fact their visualization on the screen.

Up to now, we have omitted to present the 8th step. This one establishes a link between the specification of the interface and of the application. Indeed, as the functional messages (related to task actions) and the interactive messages are based on the same actions and objects they can be put into correspondence. This step is useful to verify if the contents of all the interactive messages proposed by the interface to an analyst has a correspondent in functional messages of both the application functions and auxiliary functions and conversely. In order to facilitate our process, we assume that there exists a biunivocal correspondence between the functional and interactive messages. However, let's remark that the control interactive messages have no correspondent in the application functions .

Up to now, we have presented a particular way to chain the different steps of the specification process. Nonetheless, it can be underlined that another chaining of these stages may also be envisaged. Indeed, as the interactive messages and the functional messages have the same contents, one could first derive all the functional messages associated to the application functions and to the auxilary functions and then, we will define the interactive messages directly from the highlighted functional messages. In this case, the accumulation point corresponding to the Step 8 is no more useful.

Let's now begin the detailed presentation of our specification process.

4.2. STEP 1: DETAILED STUDY OF THE ANALYST'S TASK.

As shown in Section 1.2.2, a fine determination of the analyst's task is a fundamental and basic step for the design of the corresponding interface. Indeed, it appears that many problems may occur because of an overfunctionality or underfunctionality [Shnei 87]. Of course, if the functionalities supported by the interface do not meet the analyst's goals, this one will probably feel frustrated and as a result, this person will reject the proposed interface. On the other hand, if the implemented functionalities overdrive the needs then, problems linked to the coupled growing complexity can be awaited such as an increase of the learning time.

Consequently, during this step, in order to try to draw the functionalities to implement as carefully as possible, we study the task performed by a knowledge engineer. The general characteristics of this task have already been underlined in Section 2.2. So, we do not insist much more about them.

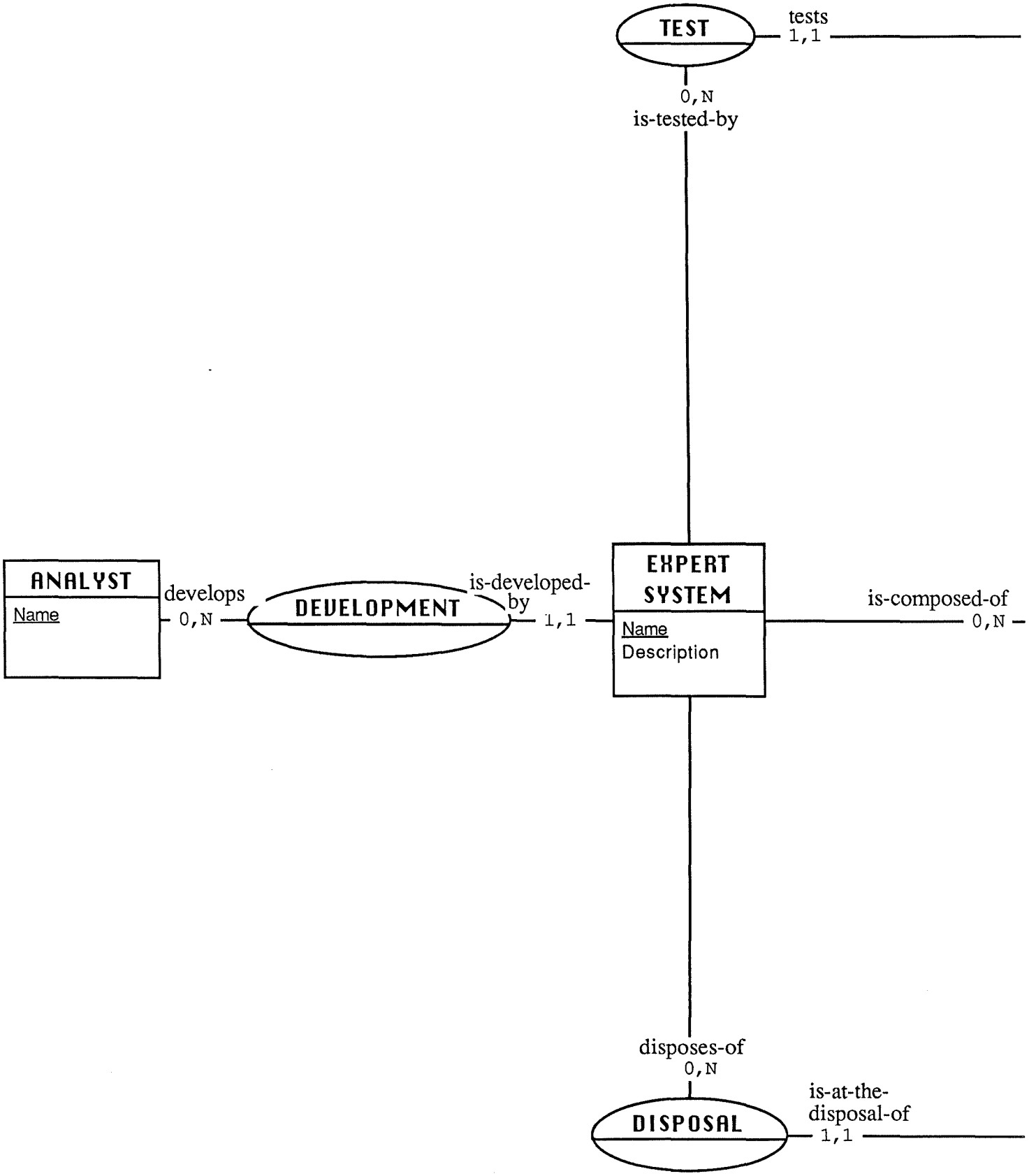
4.3. STEP 2: EXTRACTION OF THE TASK OBJECTS.

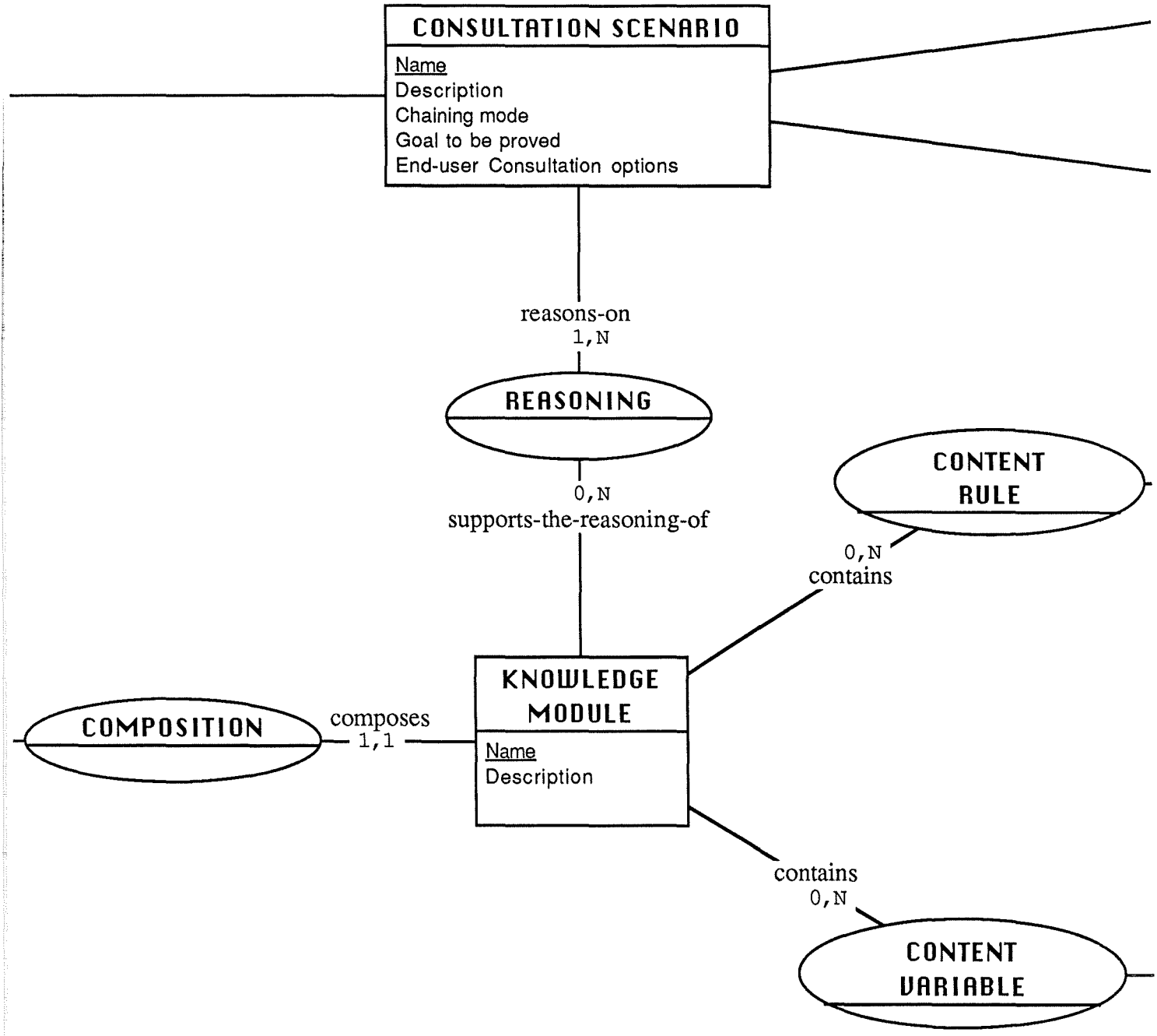
As, at this point of the process, the analyst's task is well known, all the objects He manipulates while instantiating an expert system shell can be extracted. Such objects are called task objects by reference to the Syntactic/Semantic model presented in Section 1.1.2.

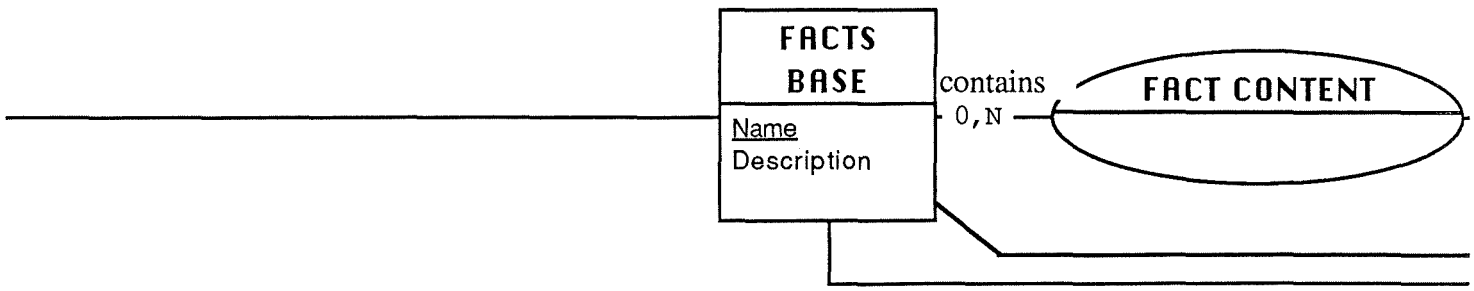
Concretely, we propose a modelization of the task objects via the Entity/Relationship formalism.

It can also be remarked that this scheme modelizes the current state of K-Expert. Indeed, for the moment, we just have to take into account the rule formalism in order to represent extracted knowledges. In the future, other formalisms such as frames will be implemented. It should be easy to integrate them in the Entity-Relationship model .

The next pages present our modelization according to the E/R formalism of the task objects manipulated by the analysts when they instantiate an expert system shell .


TEST
tests
1,1
0,N
is-tested-by
ANALYST
Name
develops
0,N
DEVELOPMENT
is-developed-by
1,1
EXPERT SYSTEM
Name
Description
is-composed-of
0,N
disposes-of
0,N
DISPOSAL
is-at-the-disposal-of
1,1


CONSULTATION SCENARIO
Name
Description
Chaining mode
Goal to be proved
End-user Consultation options
reasons-on
1,N
REASONING
0,N
supports-the-reasoning-of
CONTENT RULE
0,N
contains
KNOWLEDGE MODULE
Name
Description
COMPOSITION
composes
1,1
contains
0,N
CONTENT VARIABLE


FACTS BASE
Name
Description
contains
0,N
FACT CONTENT

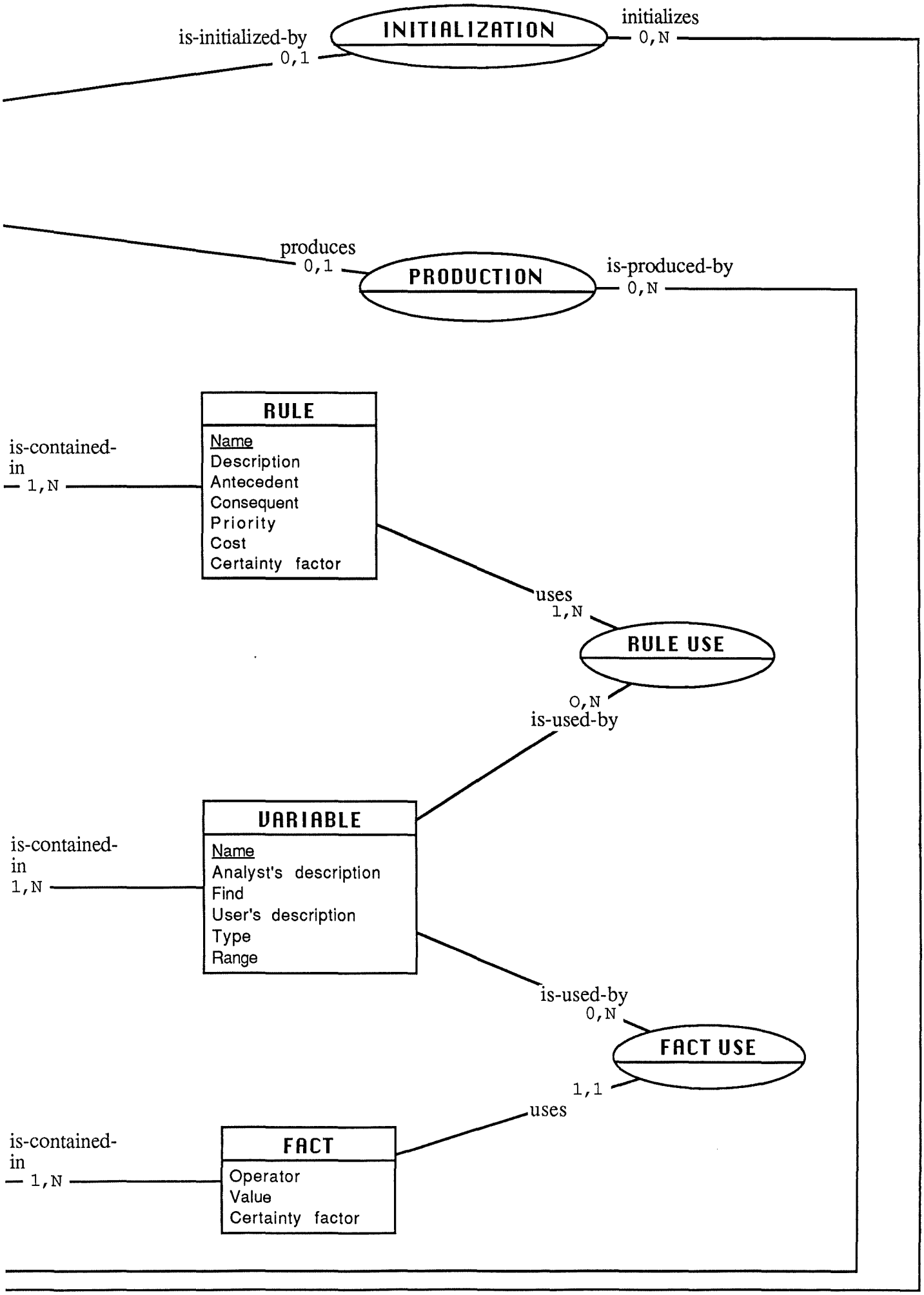


Figure 4.1 : Entity/Relationship model of task objects.

Let's now describe the stored data structures (Entity-Relationship model). We want to remark that only the significant concepts for our work are the object of this description.

◊ **Structuration of the memorized information : ANALYST**

An **ANALYST** entity

represents every person developing an expert system ,

is characterized by :
a **name**

plays the role of ***Analyst (develops)*** in none or many ***expert system*** ,

is identified by its *name* ,

Remark :

This entity is only useful if K-Expert shell is instantiated by analysts working on a main frame or with a network. In this case, it represents all the knowledge engineers who may communicate together in order to exchange expert system components they have already developed.

This option may be interesting if the instantiation of the expert system shell is led by more than one analyst.

Explanation of the connectivities :

In order to explain the *none* connectivity of the evoked role, we can say that we consider that an analyst can be mentioned in the data structures even if He has not developed an expert system with the considered expert system shell up to now.

The fact that we enable an analyst to develop *many* expert systems may be criticized. Indeed, it can seem unrealistic to develop more than one expert system instantiation at a time. However, our choice is justified by the fact that our environment is not dedicated to a specific expert system instantiation. In

our perspective, the analyst works on only one instantiation at a time. This instantiation can be a new one but it can also be selected among the already existing ones if the analyst whishes to perform its maintenance without leaving the environment. Moreover, our way to do gives the analyst the ability to display parts of already instantiated expert systems to obtain informations about them.

◊ **Structuration of the memorized information : EXPERT SYSTEM**

An **EXPERT SYSTEM** entity

represents the set of expert systems instantiations realized with the available shell;

is characterized by :

a **name**

a **description**
an explanation about the domain of the expert system and its goal ,

plays the role of ***expert system (is-tested-by)*** in none or many ***Consultation scenario*** ,

of ***expert system (is-composed-of)*** in none or many ***Knowledge module*** ,

of ***expert system (disposes-of)*** in none or many ***Facts base*** and

of ***expert system (is-developed-by)*** in one and only one ***Analyst*** ;

is identified by its *name* .

Explanation of the connectivities :

The *none* connectivity of the three first roles is explained by the fact that during its creation an expert system requires only a name and a description.

By the *many* connectivity of the first role, we express the fact that an expert system may be tested on several chaining modes, trace levels, facts bases, knowledges modules and so on, according to the considered application and end-user class.

By the *many* connectivity of the second role, we mean that an expert system can be composed of many knowledge modules. Thanks to the concept of knowledge module, we give the analyst the opportunity to structurate the whole expertise He must input into the shell around several main domains. Indeed, each module contains all the knowledges related to one main domain. So, like it happens in the Mycin expert system, a consultation may be closer to the behaviour of a particular human expert seeing that all the questions related to a subject can be asked at the same time. Consequently, the modularization helps to modelize the expert's way to process and makes a consultation more understandable for an end-user. As a result, if the analyst wishes to structurate the introduced knowledges into several domains, the corresponding expert system will be composed of many knowledge modules. Moreover, as we offer the analyst the opportunity to keep several modules, we also enable him to conserve several versions of a same module. Nonetheless, the versions management is not automatic, it may be completely supported by the analyst himself . Consequently, t is up to him to give the appropriate modules names.

We consider that a facts base contains all the data which are useful to start a consultation of an expert system. So, by the *many* connectivity associated to the third role, we enable the analyst to test the behaviour of the instantiated expert system on various pre-defined sets of tests. Indeed, a facts base may correspond to a particular test illustrating a typical case which could be treated by the expert system. By this way, the analyst has the ability to consult an expert system from a pre-defined facts base rather than having to type all the facts it requires during a consultation.

By the fourth role, we want to express the fact that an instantiation of an expert system is developed by only one analyst. If the expert system has to be built by a team of analysts, each of them works on his own instantiation. In this perspective, the team must be careful to coordinate the work performed by all its members in order to preserve the general consistency.

◊ Structuration of the memorized information : CONSULTATION SCENARIO

A **CONSULTATION SCENARIO** entity

represents a set of parameters having to be set before starting a consultation of a particular expert system;

is characterized by :

a **name**

a **description**
an explanation indicating to which users class the consultation defined is going to be proposed ,

a **chaining mode**

a **goal to be proved**
in the case of a backward chaining ,

an **End-user consultation options**
a set of options which enable the analyst to customize the end-user consultation interface according to the particular needs of a given class of them ,

plays the role of ***Consultation scenario (reasons-on)*** in one or many ***Knowledge module*** ,

of ***Consultation scenario (is-initialized-by)*** in none or one ***Facts base*** ,

of ***Consultation scenario (produces)*** in none or one ***Facts base*** and

of ***Consultation scenario (tests)*** in one and only one ***Expert system*** ;

is identified by its *name* .

Explanation of the connectivities :

By the *one* connectivity of the first role, we mean that as the aim of a scenario consists of testing at least one knowledge module, the name of one of the existing modules must be given.

Seeing that we enable an analyst to structurate his knowledge base into domains, the *many* connectivity is needed in order to give him the ability to test more than one domain of the knowledge base or even the whole base at a time. Nonetheless, this way to do could be avoided. Indeed, the best solution should be to load first a principal module in which some rules would trigger off the loading and the consultation of other modules. As K-Expert does not offer such a feature, we propose to constitute a single knowledge base by loading all the relevant knowledge modules at the beginning of a consultation. So, we conciliate the advantages related to the work with modules and the existent.

By the second role, we enable an analyst to start a consultation with a predefined set of data stored in a facts base. However, it is not mandatory to specify such an initial facts base. In this case, all the data useful for a consultation must be introduced directly by the analyst at the request of the inference engine. This explains the *none* connectivity linked to this role.

The third role explains the fact that at the end of a consultation, all the data known by the inference engine can be stored in a specified facts base, if necessary. This option could be useful in order to restart a consultation on this one. As it is not mandatory to specify such a facts base, we retain a *none* connectivity.

The *one and only one* connectivity of the fourth role is justified by the fact that a scenario contains all the attributes required to consult a particular expert system. Consequently, it is specific to a well-defined consultation.

◊ Structuration of the memorized information : KNOWLEDGE MODULE

A **KNOWLEDGE MODULE** entity

represents a human expertise modelization related to a well defined knowledge domain ;

is characterized by :

a **name**

a **description**
the explanation of the knowledge module goal ,

plays the role of ***Knowledge module (supports-the-reasoning-of)*** in none or many ***Consultation scenario*** ,

of ***Knowledge module (contains)*** in none or many ***Rule*** and

of ***Knowledge module (contains)*** in none or many ***Variable*** ,

of ***Knowledge module (composes)*** in one and only one ***Expert system*** ;

is identified by its *name* .

Explanation of the connectivities :

The *none* connectivity of the three first roles is deduced from the fact that during its creation, a knowledge module requires only a name and a description.

The *many* connectivity of the first role indicates that a knowledge module may be tested by many scenarios. This ability enables an analyst to choose which chaining mode is the most appropriate for this module, for example.

The *many* connectivity of the second and third roles indicates that this entity consists of an aggregate of the entities **RULE** and **VARIABLE**.

The *one and only one* connectivity is explained by the fact that a module is a component of an expert system. Consequently, it must always be linked to one of them. Moreover, as this module contains a part of a given expert system, it can only belong to the expert system related to the same subject than that it treats.

◊ Structuration of the memorized information : RULE

A **RULE** entity

represents a modelization of a part of the considered human expertise.;

is characterized by :

a **name**

a **description**
the explanation of the rule goal ,

a **priority**
during a consultation, when more than one rule can be fired, it can be useful to report to such a factor in order to choose the most important among them ,

a **certainty factor**
a coefficient which affects a weight to the facts deduced from this rule,

a **cost**
during a consultation, when more than one rule can be fired, it can be useful to choose the one which possesses the cheapest action ,

an **antecedent**
the condition of a rule activation (this one is defined later in this Section) ,

a **consequent**
the conclusion of a rule (this one is defined later in this Section) ,

plays the role of ***Rule (is-contained-in)*** in one or many ***Knowledge module*** and

of ***Rule (uses)*** in one or many ***Variable*** ,

is identified by its *name* .

Explanation of the connectivities :

As a rule is a part of a knowledge module, the *one* connectivity of the first role is justified by the fact that a rule must always belong to one knowledge module.

Seeing that we give the analyst the opportunity to keep more than one version of a knowledge module, the same rule can be present in more than one module. This justifies the *many* connectivity of the first role.

The *one* connectivity of the second role is linked to the fact that a rule modelizes a part of a human knowledge which relates to objects represented by variables. Consequently, at least one of them must appear in a rule.

◊ **Structuration of the memorized information : VARIABLE**

A **VARIABLE** entity

represents an instantiation of an objects class of the real world ;

is characterized by :

a **name**

a **description**
the meaning of the variable ,

a **question**
a text which is displayed during a consultation when the expert system needs more informations about this variable in order to continue its reasoning process ,

plays the role of ***Variable*** ***(is-contained-in)*** in one or many ***Knowledge module*** ,

of ***Variable*** ***(is-used-by)*** in none or many ***Rule*** and

of ***Variable*** ***(is-used-by)*** in none or many ***Fact*** ;

is identified by its *name* .

Explanation of the connectivities :

As a variable is a part of a knowledge module, the *one* connectivity of the first role is justified by the fact that a variable must always belong to one knowledge module.

The *many* connectivity, on its side, may be explained by the fact that variables are used to communicate informations between modules in which they appear.

The *none* connectivity of the second and third roles enables an analyst to define a variable before using it in a rule or in a fact.

On the other side, the *many* connectivity of the second role enables a variable to appear in many rules.

As for the *many* connectivity of the third role, we can say that more than one fact may use the same variable as we give the analyst the opportunity to affect various values to the same variable

◊ **Structuration of the memorized information : FACTS BASE**

A **FACTS BASE** entity

represents a set of facts; Let' s note that the fact notion is defined later in this section.

is characterized by :

a **name**

a **description**
the meaning of the facts base ,

plays the role of ***Facts base (initializes)*** in none or many ***Consultation scenario*** and

of ***Facts base (is-produced-by)*** in none or many ***Consultation scenario*** and

of ***Facts base (contains)*** in none or many ***Fact*** and

of ***Facts base (is-at-the-disposal-of)*** in one and only one ***expert system*** ;

is identified by its *name* .

Explanation of the connectivities :

The *none* connectivity of the two first roles is explained by the fact that a facts base can be present in the data structures without initializing any consultation scenario or without being the production of a consultation.

By the *many* connectivity of the first role, it is possible to indicate the facts base providing the initial facts for a consultation of a given expert system. The same facts base may initialize several scenarios and so, it enables the analyst to compare them in order to determine which one is the best.

By the second role, it is also possible to indicate the facts base that a particular expert system produces during its reasoning (from initial, deduced and input facts). So, the *many* connectivity of this role is justified by the fact that the same name of a produced facts base can be used by more than one scenario.

The *none* connectivity of the third role follows from the fact that during its creation, a facts base requires only a name and a description.

The *many* connectivity of this role indicates that the entity **Facts base** consists of an aggregate of the entity **Fact.**

The *one and only one* connectivity of the fourth role may be deduced from the fact that a facts base contains data useful for the consultation of an expert system. So, these data are related to the subject of this one and can not be used by another one.

◊ Structuration of the memorized information : FACT

A **FACT** entity

represents an information unit used to feed the inference engine during its reasoning process ;

is characterized by :

an **affectation operator**
the syntax of this one is defined later in this Section ,

a **constant**
the syntax of this one is defined later in this Section ,

a **certainty factor**
a number reflecting the confidence attached to this fact ,

plays the role of ***Fact (uses)*** in one and only one ***Variable*** and

of ***Fact (is-contained-in)*** one or more ***Facts base*** ;

is identified by its *affectation operator* , its *constant* and the role *uses* .

Explanation of the connectivities :

As a fact represents an information unit whereas a variable modelizes an object of the considered knowledge, a fact can use *one and only one* variable.

Seeing that a fact is a part of a facts base, the *one* connectivity of the second role is justified by the fact that a fact must always belong to one knowledge module.

As we enable the analyst to test an expert system with more than one facts base, a same fact may be contained in more than one of them. This justifies the *many* connectivity of the second role.

◊ Syntax of the rules and of the facts

◊ Fact

<fact> ::= <variable> '=' <constant>

◊ Action (consequent)

<action> ::= <action> 'AND' <action>

<action> ::= <call procedure>

<action> ::= <variable> <affectation operator> <expression>

<expression> ::= <expression> <arithmetical operator> <expression>

<expression> ::= <variable>

<expression> ::= <constant>

<expression> ::= <call function>

<affectation operator> ::= '='

<arithmetical operator> ::= '+' | '-' | '*' | '/'

A function and a procedure may consist of :

- an access to a spreadsheet, updating...etc ;
- an access to another expert system, updating...etc ;

- an access to a Data base, updating...etc ;
- the determination of a value as a result of a more or less complex algorithm ;
- and so on...

Let's note that a function always returns a result value by difference to a procedure.

◊ Boolean expression (antecedent)

<Boolean expression> ::= <Boolean expression><logical operator><Boolean expression>

<Boolean expression> ::= NOT (<Boolean expression>)

<Boolean expression> ::= (<Boolean expression>)

<Boolean expression> ::= <expression><comparison operator>< expression>

<logical operator> ::= 'AND' | 'OR' | 'XOR'

<comparison operator> ::= '<' | '>' | '≤' | '≥' | '=' | '><' | 'EQUAL' |

'UNEQUAL'

4.4. STEP 3: DEFINITION OF TASK ACTIONS.

For each of the entities highlighted thanks to the construction of the Entity-Relationship model, we are now going to define related actions.

These actions correspond to the relevant functionalities that should be offered to an analyst in order to help him to perform his job. So, during this step, we keep in mind the general features that a tool should possess in order to assist an analyst efficiently. These characteristics have been underlined in Section 2.2.2.

The expected functionalities are actions directly related to the domain of the task.

Now, let's give the list of all the actions related to the objects highlighted in the previous section.

◊ **Actions related to an analyst :**

1. COPY an expert system developed by an analyst with all its components to another one.

2. COPY a knowledge module of an expert system developed by an analyst to another one.

3. COPY a scenario of an expert system developed by an analyst to another one.

4. COPY a fact base of an expert system developed by an analyst to another one.

◊ **Actions related to an expert system :**

1. CREATE an expert system identified by a name and give its description.

2. UPDATE an expert system description and/or its name.

3. CONSULT an expert system description.

4. LIST all the existing expert system names.

5. SELECT an expert system and work on it.

6. DELETE an expert system (and all its components which are highlighted in the E-R model).

7. PRINT the name and the description of one or all the existing expert system.

◊ **Actions related to a knowledge module (KM):**

1. CREATE a KM identified by its name and give its description.

2 UPDATE a KM description and/or its name.

3 DUPLICATE a KM in order to create a new version of a module without any copying effort (name, description and all its components which are highlighted by the E/R model).

4. LIST all the existing KM.

5. SELECT a KM to work with it.

6. DELETE a KM and all its components (rules and variables).

7. PRINT the name and the description of one or all the existing KM.

8. CONTROL the consistency of a KM. By consistency, we mean the control that can be provided on the various components of a KM and on the relations between them. Among these actions, we think particularly to :

- control the existence of a definition for each variable used in one or more rules and/or in facts ;

- control if there are variables defined but never used.

This is only a first step. Indeed, the control of consistency inside a KM is currently a subject of research.

◊ **Actions related to rules :**

1. CREATE a rule and set its attributes.

2. UPDATE the attributes of a rule.

3. DELETE a rule.

4. COPY a rule or a part of it - this task is useful when the analyst intends to create a rule which is nearly the same as an existing one.

5. CONSULT a rule. The consultation opportunities are the following ones :

   - consult a rule with a given name ;
   - consult a rule containing a given variable ;
   - consult a rule whose antecedent contains a given variable ;
   - consult a rule whose consequent contains a given variable.

6. BROWSE the rules.

7. PRINT all the rules which are selected according to the consultation opportunities.

◊ **Actions related to variables :**

1. CREATE a variable and set its attributes.
2. UPDATE the attributes of a variable.
3. DELETE a variable.
4. COPY a variable.
5. CONSULT a variable of a given name.
6. BROWSE the variables.
7. PRINT all the variables.

◊ **Actions related to facts bases (FB) :**

1. CREATE a FB and give its description.
2. LIST all the existing FB.
3. SELECT a FB.
4. COPY a FB.
5. DELETE a FB.

6. UPDATE a facts base description and/or its name.

7. CONSULT a facts base description

8. PRINT the name and the description of one or of all the existing FB.

◊ **Actions related to facts**

1. CREATE a fact.

2. UPDATE the attributes of a fact.

3. DELETE a fact.

4. CONSULT a fact.

5. BROWSE the facts.

6. PRINT all the facts or only the facts containing a given variable.

◊ **Actions related to the consultation scenarios :**

1. CREATE a scenario and set its attributes.

2. UPDATE the attributes of a scenario.

3. LIST all the existing scenarios.

4. SELECT a scenario.

5. CONSULT the expert system with a selected scenario.

6. DEBUG the consultation of an expert system with a given scenario.

7. DELETE a scenario.

8. COPY a scenario.

9. PRINT one or all the existing scenarios.

4.5. STEP 4: DEFINITION OF COMPUTER OBJECTS AND ACTIONS.

The functionalities highlighted by the Entity-Relationship model of the analyst's tasks has only provided task actions [Shnei 87]. However, it seems that it could useful to join them other possibilities in order to support the analyst while He is performing task actions. Indeed, as the analyst has to interact with a computer, we must also take into account objects related to the computer world (e.g. the computer objects [Shnei 87] which have been evoked in Section 1.1.2) and specify the actions that apply to these objects (e.g. the computer actions). We can also identify this computer actions as task actions.

Among the computer objects, we retain :

- The "Session concept". Indeed as the analyst will develop an expert system instantiation with a computer which is in fact a volatile environment, it is useful to give him the opportunity to leave his work whenever He wants and to retrieve it in the same state later. In this perspective, we call "Session" the storage of the whole interaction between the analyst and the computer from the starting of his work up to the moment He stops working. The actions associated to this concept are :
    1. SAVE the current session state under a given name. It means the screen configuration (e.g. opened windows, their size, contents and position), the set parameters...etc.
    2. LIST all the existing sessions.
    3. RESTORE a given session.
    4. DELETE a session.
- The "environment parameters" concept. The analyst has particular needs and particular computer abilities. So, it is interesting to give him the ability to customize the used computer environment according to them. Consequently, it is necessary to feature the environment by many parameters such as the used printer, default values that must be considered if the analyst does not type any other value...etc. The action that can be applied to this computer object is:
    1. SET the parameters to the desired values.

4.6. STEP 5 : DEFINITION OF FUNCTIONS SUPPORTING TASK ACTIONS.

The aim of this step is to extract a set of functions which should be implemented in order to support all the task actions deduced in the two previous sections.

All these functions are described by the mean of the model of the static of processes.

This model is particularly useful for us because relying on it, it is possible to determine all the input and output informations tied up to each considered function. This gives us the ability to introduce the notion of message and to be more precise of functional message.

This type of message plays a fundamental role in the specification of an interactive application because input functional messages describe the informations that should be introduced by the analyst to perform a function whereas the output functional messages indicate the information that should be returned to the analyst at the conclusion of the realization of a function. So, we have to link a function to each task action (e.g. functionality). Moreover, it can be remarked that the contents of a functional message can be deduced from the attributes linked to the object associated to the considered action. These attributes are those described in the Entity/Relationship model.

Let's illustrate the use of the model for the functionality "create a rule and set its attributes". The corresponding function is named "create-rule".

Example : Model of the static of processes.

Function : CREATE-RULE

Objective : To record a new rule in a selected knowledge module of an expert system.

Input functional message :

create-rule-input : this message presents the attributes associated to a rule in the Entity-Relationship model :

- rule-name ;
- rule-description ;
- rule-antecedent ;
- rule-consequent ;
- rule-priority ;
- rule-cost ;
- rule-certainty.

Pre-conditions :

The given rule-name does not correspond to an existing one.

The given rule-antecedent and consequent are syntactically correct.

The given priority, cost and certainty vary between two pre-defined constant values.

An expert system and a knowledge base module have already been selected. (This condition is useful in order to respect the objective of the function).

Output functional message :

create-rule-ok : This message indicates that the given rule has been added to the wished module ;

create-rule-nok : This message indicates that the given rule has not been added to the desired module.

4.7. STEP 6 : DEFINITION OF AUXILIARY FUNCTIONS.

The use of the model of static of processes during the previous step of the specification process enables us to deduce new functions which are not directly linked to the task of the analyst but which must be offered by the interface in order to realize the considered functionalities. Indeed, by

emphasizing on the input messages of the function corresponding to a functionality, some pre-conditions that the input information elements have to respect in order to enable the function to perform its work correctly may be highlighted. So, some additional functions should be added in such a way that these pre-conditions can be established before executing the function corresponding to the functionality itself.

As a result, in order to satisfy the conditions which have been linked to the "create-rule" function, the following functions have to be implemented :

- verify-rule-name-exist ;
- verify-antecedent-syntax ;
- verify-consequent-syntax ;
- verify-priority ;
- verify-cost ;
- verify-certainty.

The "select-expert system" and "select-knowledge-module" functions correspond to two functionalities highlighted in Section 4.4. .

These auxiliary functions are also described by the model of the static of processes.

Let's now present the static of the "verify-rule-name-exist " function.

Function : VERIFY-RULE-NAME-EXIST

Objective : To consult the already recorded rules of a related knowledge module of an expert system in order to find if the given rule-name already exists.

Input functional message :

verify-rule-name-exist: This message presents

- rule-name ;

-expert-system-name ;

-knowledge-module.

Pre-conditions :

The given expert-system-name exists.

The given knowledge-module-name exists for the given expert system.

Output functional message :

verify-rule-name-exist-ok : The contents of this message indicates that the rule name already exists ;

verify-rule-name-exist-nok : The contents of this message indicates that the rule name does not already exist.

4.8. STEP 7 : EXTRACTION OF INTERACTIVE MESSAGES.

Up to now, we have defined the data structures, all the necessary functionalities and also the functions that must be supported by the application in order to provide these functionalities. The next step consists of defining the dialogue. The aim of this dialogue is to support the interaction between the analyst and the interactive application in order to realize the pre-defined functionalities.

First of all, let's introduce a new concept that will play a fundamental role in the remaining of this exposure. The considered concept is that of interactive message. Each interactive message embodies all input or output datawhich are meaningful for the user. Consequently, all the constitutive informations of such a message have to be semantically linked and moreover, the retained regrouping must be significant for the user from an ergonomic point of view.

As these functionalities are actions related to task objects, the contents of an interactive message consists generally of all the attributes linked to the same object.

Before presenting concretely some of the interactive messages of our application, it seems necessary to describe interactive messages in a rather systematic way. So, we propose to do it by defining the following elements for each of them. This description is inspired by [War 88c], [War 88b].

Interactive message description :

- Name ;
- Definition explaining its object ;
- Type (input, output, control, error or help) ;
- Data it concerns and the related integrity constraints ;

- Justification : It means an exploration of the reasons for which this message can be perceived as a dialogue unit for the user ;
- Operations that analysts will be able to perform on the message.

According to the message type, various standard operations may be defined. Seeing their standard character, these operations will be omitted during the message description.

Indeed, *input interactive messages* and *control messages* may always be submitted to the following operations:

- Create an occurrence ;
- Suppress an occurrence ;
- Put away an occurrence (for example by iconizing it);
- Select a put away occurrence ;
- Conclude an occurrence. It means to validate its contents and to trigger off the actions associated to its conclusion.

Moreover, each field of such a message may be submitted to :

- Affectation of a value ;
- Suppression of a value ;
- Updating of a value.

From another side, *output interactive messages* are concerned by the following operations :

- Put away an occurrence ;
- Select a put away occurrence ;
- Conclude an occurrence. It means to erase the corresponding message from the screen and to switch to the next interactive message.

As to the *help messages* , it is only possible to perform the following actions on them:

- Create an occurrence ;
- Conclude an occurrence.

We are now going to propose an example of this process for some significant functionalities we have illustrated in Section 4.5. .

Name : CREATE-RULE-INPUT

Definition : Set of the attributes composing a new rule. It corresponds to the functional message CREATE-RULE-INPUT presented in Section 4.6. .

Type : Input interactive message.

Data : rule-name ;

rule-description ;

rule-antecedent ;

rule-consequent ;

rule-priority ;

rule-cost ;

rule-certainty factor.

The syntax of these fields will have to be defined when physical implementation of this message will be considered.

Justification : The contents of this message corresponds to all the values necessary to input a new rule.

Operations : All the standard operations associated to input messages. Moreover, there are syntactic controls on characters introduced in each field and semantic control if needed.

Name : SELECT-EXPERT-SYSTEM-INPUT

Definition : Input of the name of an expert system to select. It corresponds to the functional message of the same name.

Type : Input interactive message.

Data : expert-system-name

Justification : This message is useful to input the information necessary for the identification of a particular expert system.

Operations : All the standard operations associated to input messages. Moreover, there is a syntactic control on characters introduced in the expert system name field and a semantic control to verify if the expert system-name exists.

Name : UPDATE-RULE-INPUT

Definition : This message makes possible the modification of a rule which has been stored in the knowledge module. It corresponds to the UPDATE-RULE-INPUT functional message.

Type : Input interactive message.

Data : Same data as those described in the CREATE-RULE-INPUT interactive message.

Justification : The contents of this message corresponds to all the values necessary to record the modified rule in the knowledge module.

Operations : The operations are the same as those performed on the CREATE-RULE-INPUT interactive message.

4.9. STEP 8 : CORRESPONDENCE BETWEEN INTERACTIVE AND FUNCTIONAL MESSAGES.

As the interactive messages and the already evoked functional messages are derived from the same functionalities and are linked to the same objects, we can mention that their contents have to be identical. Let's recall the already mentionned hypothesis concerning the biunivocal correspondence between these two kinds of messages.

Moreover, the interactive application needs some other interactive messages which may not correspond to functional messages (such as help, error and control messages). Indeed, while designing the interactive messages, one may realize that some informations necessary for the user have no counterpart in functional messages. This follows from the fact that interactive messages are defined by reference to the user's needs in information while functional messages are dragged from the functionalities that should be implemented.

During this step, the functional messages are put into correspondence with interactive messages. This is useful to verify if no message has been forgotten.

In our particular case, this step is already realized. Let's note that if an interactive message corresponds to a described functional message, they have the same name. This option avoids increasing the number of names to handle.

4.10. STEP 9 : BUILDING OF THE CONVERSATION SCHEME.

This step has been inspired from the conversation scheme defined by [War 88b].

The interactive messages are also submitted to chaining rules. Indeed in order to have a consistent dialogue, the input and the output of interactive messages may not occur at any time, they must respect chaining rules describing the actions authorized at any time during a man-machine interaction. So a conversation scheme must be specified to modelize the dialogue evolution.

The formalism of this scheme is close to this of the dynamic model. However, it considers only interactive messages, so it does not make appear treatments. Indeed, conversely to what happens in the dynamic of treatments, the functions that are performed on interactive messages are standard operations (as input, output, and so on).

Starting from the extracted interactive input-output messages, we can now define a conversation scheme corresponding to the chaining constraints that may be applied on them. We can visualize the conversation corresponding to our application on the following Figure 4.2.

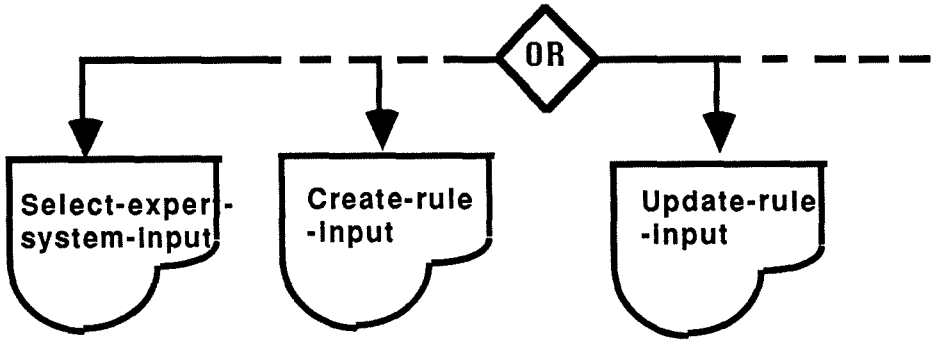


Figure 4.2 : Conversation scheme.

The scheme we propose starts with a conditional structure. This one is represented by a lozenge. Indeed, in our case, the analyst is completely free to undertake every action He wants. So, He can either add a new rule with its corresponding interactive message (create-rule-input) or modify an existing rule (modify-rule-input) or perform one of the previously described functionalities as soon as He has launched K-Expert. This follows from the fact that we have decided to consider our interactive application as a toolbox of actions.

4.11. STEP 10 : EXTRACTION OF CONTROL INTERACTIVE MESSAGES

From the study of the conversation scheme, one can deduce so-called control messages.

The "control messages" are particular interactive messages which are used to orientate the dialogue in process. They enable the user to perform choices among alternatives during a man-machine interaction session. In a word, they correspond to decision points.

In our conversation scheme, we can see an "or" structure. This structure gives the analyst the ability to choose one of the functionalities of the application. A control message has to correspond to this structure in order to enable the analyst to perform his choice. We can specify it in the following way.

Name : CHOOSE-FUNCTIONALITY

Definition : This message enables the analyst to choose the desired functionality among those offered by the application .

Type : Input interactive message, control message.

Data : A field which takes a value corresponding to the chosen functionality.

Justification : The contents of this message corresponds to the only data needed to indicate the wished choice.

Operations : All the standard operations that are associated to input messages.

At this point of the work, we have completed the specification of all the concepts related to the task to interface. We must now take into account the analyst's profile in order to justify our ergonomic options. This point is treated in the next Section.

4.12. STEP 11 : TAKING INTO ACCOUNT OF THE ANALYST'S PROFILE

Starting from the analysis of the profile of a typical analyst that has been realized during Chapter 2, we must now draw general ergonomic options. The latters consist of the definition of guidelines that should be kept in mind during the next step in order to build an interface that is satisfactory from an ergonomic point of view.

In this perspective, we try to put the effort of crossing the evaluation and execution gulfs [Norm 86] on the interface designer rather than on the end-user.

Concretely speaking, what can be deduced from Chapter 2 is that :

- Analysts of expert system perform an intellectual and creative job with a high level of motivation. This level is as high as their requirements about the interface [Shnei 87]. As a result, the proposed interface can not be an elementary one.

- They are often experts in the so-called task-domain but their knowledge of the computer-concepts [Shnei 87] is not necessarily very wide. Consequently, the proposed interface should give them the illusion to work directly with relevant task-objects. In the same way, this option would enable them to concentrate themselves on the task-itself.

- Finally as the considered analysts' spectrum stretches from novices to experts, it could be fine that the interface be evolutionary.

Concretely, we try to stay as sensitive as possible to the notion of "user friendliness". This fundamental concept is not an abstract one but is defined concretely by Shneiderman [Shnei 87] by five criteria which are the learning time, the retention over time, the rate of use errors, the level of subjective satisfaction and the speed of performance. We intend to try to optimize all these elements while designing the interface.

Moreover, we think it is particularly necessary for the interface to provide the analyst a feeling of control on his job and also to encourage him to explore its abilities.

4.13. STEP 12 : EXTRACTION OF INTERACTIVE OBJECTS.

The aim of this section corresponds to the visualization of the analyst's interface.

The proposal of concrete interactive objects supporting the implementation of the functionalities we have extracted will be the last point of our interface design. The interactive objects are the physical representations of one or more interactive messages.

To define interactive objects, we have accomplished a semantical grouping of interactive messages. Indeed, we put into a single interactive object all the interactive messages related to a same information structure (for

example, all the fields corresponding to a rule are presented through a single mask) and also all the actions (functionalities) an analyst may perform on such an information structure (for example, all the operations an analyst is authorized to perform on a rule such as create it, modify it, delete it ...are accessible from the same interactive object).

This last step relies on the previous sections of this chapter. Indeed, we try to conceive interactive objects in such a way that they represent all the available task actions (e.g. : they correspond to the highlighted interactive messages). We also take into account the constraints that may have been discovered and the ergonomic considerations that have been underlined in Section 4.12.

But first of all, before introducing a description of the basic standard interactive objects we instantiate to give a physical form to the interactive objects constitutive of the interface, let's evoke a concretization of the ergonomic guidelines deduced in Section 4.12. This one is reflected in all the instantiations of generic interactive objects that we provide in Section 4.13.3. .

4.13.1 Ergonomic options

The presentation of the ergonomic options we have retained are structurated into various sections corresponding to the criteria we have selected in order to criticize existing expert systems shells in Chapter 3. To be complete in this section on which our proposed interactive objects rely deeply, we have added some additional criteria which are those defining the user friendliness [Shnei 87]. This concept introduced by Shneiderman has been presented in Section 4.12. .

Let's start the exposure of each of these sections.

4.13.1.1. Control feeling and initiative

At the end of reading texts as those written by Norman [Norm 86], we think that a transparent interface (e.g. an interface in which an analyst may think He is directly accomplishing the task) based on a world metaphor rather than on a "conversation metaphor" is the most appropriate one. So, we intend to give the analyst the control of the interaction as much as possible. To satisfy this objective, we propose to define an initial window displayed at the

beginning of the analyst's work and enabling him to access to all the basic development functionalities. This initial window can not be closed so that the analyst does not risk to be lost. Moreover, many paths are usable to reach the same action. This option is also interesting here because except for the first window, the analyst may initiate everything He wants in the desired order.

Moreover, we base our analyst's interface on a multi-windowing system. This seems particularly interesting because it corresponds to human thinking way as it has been showed in Section 1.1.3.

Each window contains a semantical unit, it means a set of semantically related informations such as a mask for rules editing and the associated actions. So, the analyst has the opportunity to work on one window and to consult other ones in order to fetch complementary helpful informations. The analyst is free to resize, to move, to close, to activate windows as He wants. Besides, when the analyst is required to fill many fields, we let him free to switch between them by using arrow keys or mouse-clicks.

A rational limit to his control on the system is the fact that He must answer to displayed dialogue boxes before being able to perform something else. Another rational limit is that the analyst is only allowed to open one and only one instantiation of each of the proposed windows at a time.

We reinforce the control of feeling by enabling the analyst to access to a pop-up containing the list of all the already activated windows. This is illustrated in Section 4.13.2.7. This is particularly interesting seeing that nothing prevents windows overlapping in a multi-windows system.

According to us, another important point is to give the analyst the ability to adapt the interface to his needs as much as possible. This can be done by the setting of a set of parameters relative to the default options applicable when the developing environment interface is started such as the selection of a printer.

Finally, let's say that we give the analyst the opportunity to leave the standard K-Expert interface in order to switch to a command language window. Seeing the advantages of this approach that have been underlined in Section 1.2.3.3., we intend to make our interface attractive for novices but

also for experts. Let's remark that the definition of such a language is beyond the scope of this work.

4.13.1.2. Memory load and internal flexibility

Let's remark that this point corresponds to the criteria "Learning process" and "Retention over time" of the previously evoked principle of user friendliness.

The theoretical section of our work has led us to consider that the less an interface user has to provide an effort to cross the gulfs separating his view of the task and the system representation, the quickest his understanding and assimilation of a system interface is. Our point of view is to assign the designer the "gulf bridging" effort [Norm 86]. Concretely, we try to design the analyst's interface in such a way that it reflects directly tasks objects on which the analyst may perform significant task actions. It is why we have made a so-called task analysis to extract significant objects and associated functionalities. This study has been presented in Section 4.1.

By reference to the syntactic-semantic model [Shnei 87], we isolate task concepts (such as the notion of a rule) and task actions (such as the creation of a rule). We try to hide as much as possible syntactic elements and also computer concepts and actions. So, for example, the analyst must never ask to save (e.g. a computer action) a rule, He just accepts it. This way to do should reduce the learning time and effort and also the required computer qualification of the analyst. To go deeper into this direction, we are also careful of the presentation of these task objects and actions to the analyst.

Whenever it is possible, we provide a graphic representation of objects and actions. For example, a rule is represented through a mask and actions associated to the rule network are symbolized by significant pictures. When a graphic presentation is not appropriate, we choose a significant word to designate the corresponding action such as the word NEW to indicate that a new rule is going to be created.

By this way, we hope to reduce the learning effort because every element is directly significant. At the same time, we intend to facilitate the retention of the interface manipulation over time.

Retention over time is also improved by the fact that we regroup semantically related objects and actions. This orientation is reflected notably by the fact that all the actions linked to a general functionality of the K-Expert development environment are put together. It is why at the starting of K-Expert, we decide to display eight icons which give access to a set of related actions. For example, the so-called editor icon enables the analyst to access to all the editors usable to introduce knowledges in the expert system shell.

To manipulate the retained objects and actions, the analyst has just to use the mouse. This mouse approach is significant for the novice and intermittent analysts. However, it can be annoying and time wasting for the experts. So we have decided to take into account the possibility to integrate a command language. The advantages (and limitations) of this interaction style have already been explained in Section 1.2.3.3. The possibility to use such a language should support the switching between a novice and an expert state.

To facilitate this progression, we implement also many ways to reach the same effect. Indeed, conservations that we have led with some expert analysts have shown that even if this internal flexibility may be slightly confusing at the beginning, it becomes soon very useful.

We think that the graphic and semantic presentation of objects and actions and also the mouse manipulation should contribute to alleviate the analyst's memory load. According to the model of the human processor presented in Section 1.1.2, this is a non negligible advantage.

Another remark to do is that the learning process is encouraged by the fact that we intend to display a confirmation dialogue box before executing potentially disastrous actions such as the deletion of a knowledge module. In the same way, each action accomplished on the contents of a window may be UNDOne. So, normally, the analyst should not be stressed to initiate actions.

4.13.1.3. Feedback

We propose to conceive our interface in such a way that the result of each initiated action is immediately reflected on the screen contents or on the cursor shape. Indeed, we consider that immediate feedback is an important quality requirement. For example, when the analysts provides the name and description of a new expert system, this one is automatically added to the

displayed list of existing expert system. Whenever an action may affect the contents of an opened window, this one is directly updated even if it is not activated.

4.13.1.4. Errors handling

Our basic principle may be stated as follows : Better to prevent errors than to correct them. So we have decided to design a mouse-use oriented interface.

Of course, as it has been highlighted in Section 1.2.3.1. this way to do may be constraining in the sense that it forces the analyst to leave the keyboard. However, there is a non negligible advantage linked to the mouse. It prevents analysts from typing errors like it can happen for the use of a command language. In fact, it reduces the use of the keyboard to fill text fields because the analyst has the ability to select an appropriate value in a list by using a mouse.

However, we let the analyst free to type directly value in fields with the keyboard if He wishes it. In this case, whenever a syntactic or semantic error occurs a message is displayed in a dialogue box. The message texts are worded in such a way that they provide a diagnostic and also indications about the way to solve the detected problem. They do not present error numbers which would send back the analyst to a reference book.

To be precise, we can say that by semantic error we mean errors such as an attempt to load a non existing knowledge module.

To be consistent with what we have learned through literature, we propose to implement the semantic and syntactic controls on analyst's inputs as early as possible.

So, in the case of an input mask composed of several fields, for example, syntactic controls are not started after the filling of all the fields but after each field filling. By this way, the analyst may react instantaneously by correcting the bad field from the input value which remains displayed.

Moreover, as we have seen for the control feeling, we offer an UNDO function. This one enables the analyst to avoid errors which could be linked to awkwardness.

4.13.1.5. On-line help

To assist the analyst during his learning period or using time, we propose semantic and syntactic help functions. At any time, a semantic information may be accessed to explain the contents of a window and of any clickable element. For any field to fill, a syntactic help message is foreseen and if there exists a list of possible values for it , this list is displayed to the analyst who can select one of them with the mouse. The concretization of this on-line help is presented in Section 4.13.3.8. .

Theoretical readings having attracted our attention on the importance of this type of support, we think the contents of these messages should be carefully designed in order to be self-sufficient (e.g. they must no send back the analyst to a reference book).

4.13.1.6. Interruption

The analyst has always the opportunity to access to the Dos system during a working session with the interface without having to leave it.

Moreover, we propose to take into account the ability to save and restore a current work session state so that the analyst is not system-dependent but can leave it at any time.

4.13.1.7. Consistency

An important hint retained to facilitate the learning process is the respect of consistency thorough the whole interface. The notion of consistency, presented in Section 1.3., permits the analyst to build an uniform view of the whole interaction. Let's remark that it should alleviate the learning process. To implement it , we propose to build all the elements of the interface such as windows, dialogue boxes and buttons by instantiation of standard ones defined in Section 4.3.1. By this way, the constitutive elements of a window, for example, will always appear at the same position. Moreover, the same basic actions (such as the moving of a window, the starting of an action

by clicking on a button...etc) are always attached to instantiations of a standard object. So this improves the analyst's feeling of security. He should never be surprised by the result of an action.

For the manipulation syntax, it emerges from literature that the best sequence is often object selection/action selection. So, we try to respect this principle as much as possible. For example to modify a rule in a rule mask, the first thing to do is to access to the desired rule (e.g. object selection) and then to click on the action MODIFY in order to access to the mask contents.

To conclude these remarks about consistency, we can also say that we always try to set the access to actions at the same place and under the same formalism. So actions giving the opportunity to modify a window contents by oneself or giving access to action menus are always set above the window and have the form of rectangle buttons. From another point of view, actions which open a dialogue box or a window are always located at the bottom of the considered window where they appear as icons.

4.13.1.8. Performance speed

Naturally, we intend to reduce the time required to perform a particular task by using the designed interface. In this perspective, we take options such as the saving of fields contents, in order to restore them when the window presenting them is reopened. Moreover, the analyst may move between mask fields or buttons using the keyboard arrows instead of the mouse. So, He must not leave the keyboard. We also define default action buttons. By this way, the analyst can replace a click on a button by a single press on the Return key. The already evoked ability to select a value in a list instead of typing it may accelerate the input rate. Finally, we arrange the proposed menus and buttons in such a way that the more frequent actions are put in front of them.

4.13.1.9 Satisfaction level

This factor seems to be one of the most important in order to design an interface that will be really used. According to Laurel [Norm 86], we must conceive an "enjoyable" system. So by reference to readings and to our personal experience of existing expert systems shells interfaces, we have

extracted the elements which are the most enjoyable for us. These elements are graphical supports such as the use of significant icons.

Besides, we think that all the factors detailed previously contribute to the growing of the satisfaction. Moreover, we try to be especially attentive to the aesthetic aspect of the screens presented to the analyst in order to not overload them.

To say more about the satisfaction linked to the use of an interface, we should of course implement a prototype of it and present it to various classes of analysts. This step is particularly important as it has been underlined in Section 1.4 but it oversteps the scope of our thesis.

Keeping in mind these ergonomic options, we can now define the standard interactive objects we consider. Then, we propose instantiations of them in order to visualize our analyst's proposal.

4.13.2 Description of used standard interactive objects.

The implementation of the considered interface relies on a User Interface Management System (UIMS) which offers the interface programmer basic standard interactive objects. These interactive objects are generic (or standard) dialogue units such as windows, pop-up, icons which enable an user to interact with the application. They must be instantiated in order to present the analyst a visualization of his task. The presentation envisaged here is useful to define the standard interactive objects which should be supported by the chosen UIMS for the implementation of the interface.

Seeing that the standard actions linked to their manipulation (like the closing, the scrolling or the resizing of a window) become well-known by an increasing number of potential users, we do not intend to present them here. Let's note that the standard objects on which our interface relies are the window, the dialogue box, the pop-up, the text entry field, the check box, the radio button, the list box, the button and the icon. To have access to more details about them, the interested reader may have a look on books or documents such as [War 88c].

In this Section, we focus only on objects and actions which are not provided directly by the UIMS. After this, we add some general principles about the use of the mouse.

4.13.2.1 The help and undo boxes

We describe only two objects that can be found in the standard window of our interface because they are not usual. There are the help box and the undo box which are presented on the following Figure 4.3.

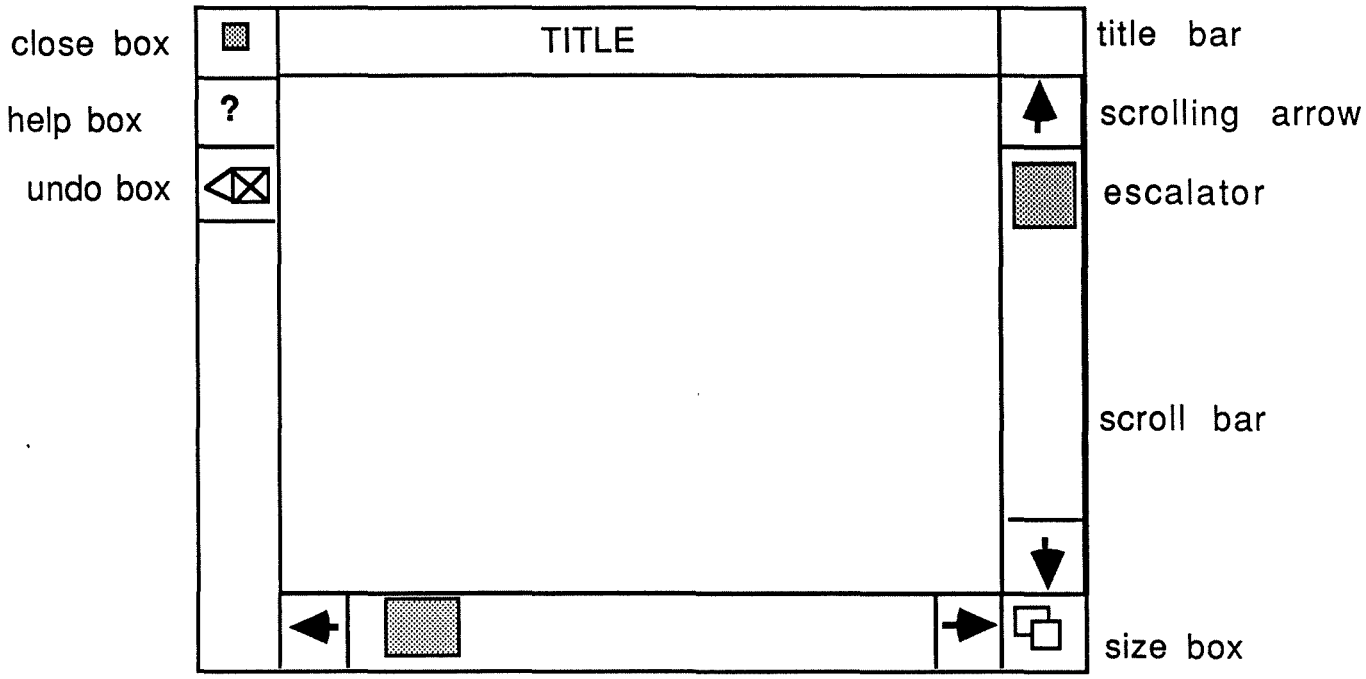


Figure 4.3: Standard window.

The **UNDO** box : cancels the last operation performed on the contents of the associated window.

The **HELP** box : displays informations about the currently activated window and its contents in order to assist the analyst while He is performing a given task.

4.13.2.2. The list icon

To input a value into a text field, the analyst has to type the desired value and to press on the Return key or to click on another object. However,

there is another way to input a value. When a field can take a value belonging to a specified set of values, the analyst can directly select the desired one in a list. In this case, the input field is followed by an icon called "list icon". By clicking on it, a pop-up is displayed in which the analyst can make a selection with the mouse. The selected value of this list is reflected in the corresponding entry field. The list icon is shown on the following Figure 4.4.

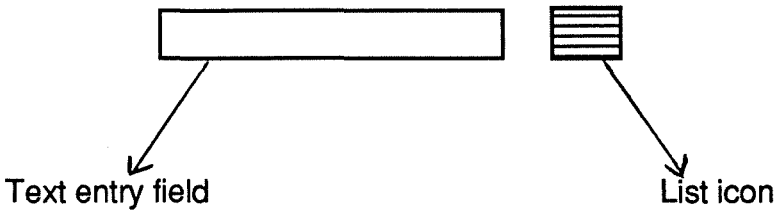


Figure 4.4 : The list icon

4.13.3.3. The flipping mechanism

When a window may present several instances of a given object, but can only display one at a time, this mechanism enables the analyst to select one of them by browsing them page by page. This may be visualized on Figure 4.5.

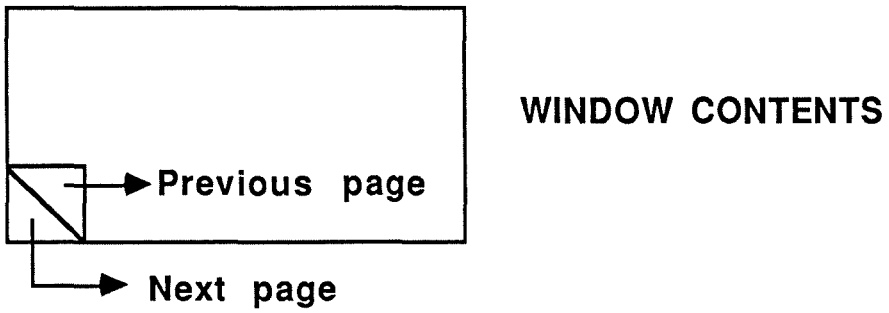


Figure 4.5 : Flipping mechanism.

Indeed, the window appears like a page whose the left corner is folded. To turn the pages, a simple click on the adequate corner is necessary.

Let's note that the considered instancies form a ring sorted alphabetically on the instancies names. When the last instance is reached, the next one corresponds to the first one.

4.13.2.4. The index mechanism

This mechanism is useful in the same context as the flipping mechanism, but it enables a quicker selection. It is illustrated by Figure 4.6.

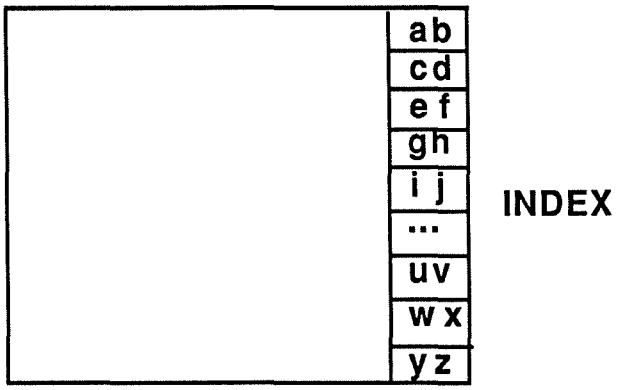


Figure 4.6 : The index mechanism.

Indeed, an index is displayed on the right side of the window. By clicking on a letter, the analyst selects the first instance starting with this letter in the alphabetical order. If there is more than one instance for the chosen letter, He can access to the desired one by using the flipping mechanism.

4.13.2.5 The buttons

Buttons are little objects offering an access to a functionality as a result of a mouse click.. In our proposal, we consider two kinds of buttons which depend on the type of interactive object to which they belong.

In dialogue boxes , buttons are rectangles with round corners displayed at the bottom.

Figure 4.7 : Button in a dialogue box.

In windows, buttons are rectangles displayed at the top.

Figure 4.8 : Button in a window.

Those buttons enable the analyst to interact directly on the window contents or to access to a pop-up if more information is needed. Let's remark that default buttons are admitted. The background of such buttons is set into a darker color to distinguish them from the other ones.

4.13.2.6 The icons

Icons are graphical representations of a function displayed at the bottom of a window. When the analyst clicks on an icon, the represented action is performed on a selected object. The icons considered are the following ones :

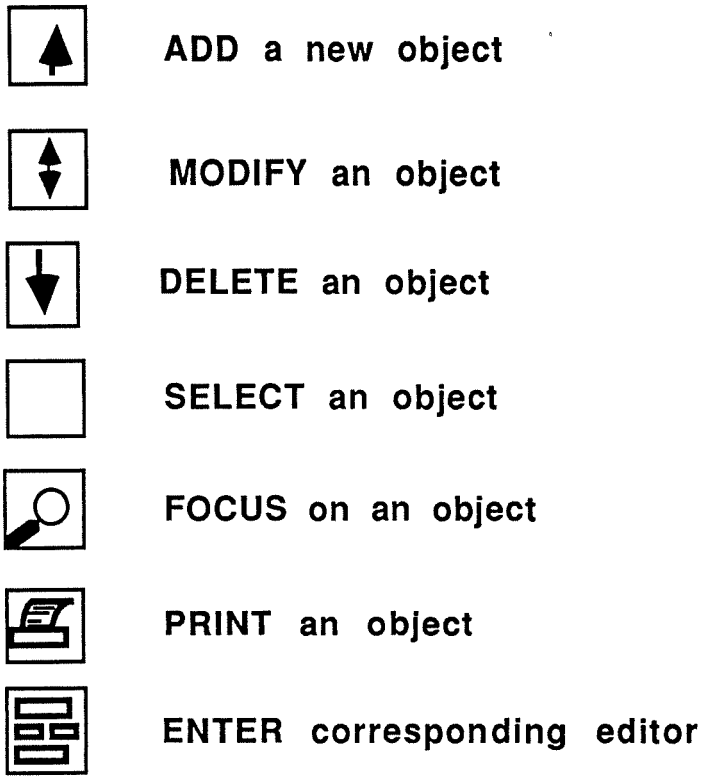


Figure 4.9. : Icons visualization.

When selected, each of these icons opens a dialogue box to enable the user to complete his command or a window if needed. There are other icons which are used only in the rule network. Their description will be given when the rule network window will be explained in Section 4.13.3.4.12. .

Moreover, the main window of the K-Expert analyst's interface contains eight icons which are always accessible to initiate a sequence of actions. They will also be described when the K-Expert window will be explained in Section 4.13.3.1. .

4.13.2.7. Mouse use

The considered mouse offers two buttons which will be designated by RB (for "Right Button") and LB (for "Left Button") in the remaining of this exposure. The functionalities linked to these buttons are permanent.

- The function of the LB corresponds to the classic one. Indeed by clicking one time on it, it is possible :

  . to select an object (which becomes highlighted as a result) on the screen ;

  . to start an action (selection of a menu item for example) ;

  . to move an object from one place to another one.

- The function of the RB consists of :

  . the display of an on-line help. If the analyst clicks on the RB when the cursor is positioned on an screen element, a help box containing syntactic informations about the corresponding element is displayed (if some help is available) ;

  . the display of a pop-up containing a list of all the opened windows. To make it appear, the analyst has just to click anywhere on the screen, the result of this click is the display of a pop-up like the following one which disappears as soon as He releases the RB:

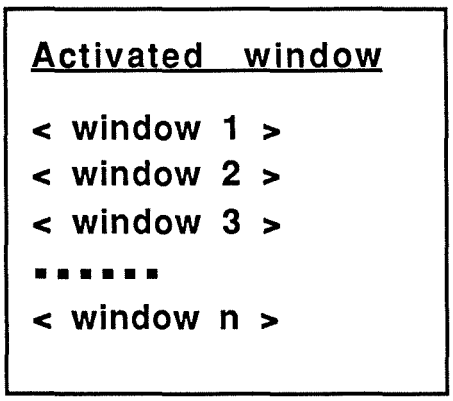


Figure 4.10 : Activated windows pop-up.

Justification of the existence of this pop-up.

- To give the analyst the ability to have a global view of his work at the time being. So we think that He would have the feeling to dominate the system (and not the contrary) because even hidden windows will appear in the list.

- To enable the analyst to reactivate directly an already opened window from anywhere. Indeed, the analyst only needs to click on the desired <window name> item with the LB to activate it automatically.

4.13.3. Specifications of the instantiations of the basic standard interactive objects.

In order to preserve the understandability and the consistency of this specification, each interactive object is defined according to the following pattern :

- name (of the considered interactive objet)

- definition

- list of the functionalities realized thanks to the implementation of this object.

- justification of the contents of this object.

- presentation : graphical presentation of the considered object.

- the actions that can be performed through the interactive object and their effects on the application chaining. As the standard manipulation of the instantiated interactive objects has already been described in Section 4.13.2., we do not mention again actions like re-sizing or moving a window.

Let's start with the main window of our interface (e. g. : the window that is displayed whenever an analyst will begin to work with K_Expert). This window remains displayed up to this person asks to leave the K_Expert interface. Moreover, it should be underlined that all the functionalities proposed to the analyst are accessible through this window.

4.13.3.1. Presentation of the main window of our interface.

Name: K-Expert-window.

Definition: Main window of the K-Expert interface giving access to all the implemented functionalities. This window presents the analyst the control interactive message "choose-functionality".

Functionalities: K-Expert window does not implement any particular functionality but it gives access to each of them.

Justification: As this screen corresponds to a single interactive message, we do not have to justify a grouping of messages. However, we can justify its particular organization (e.g. : eight icons meaningful for the analyst and under which ones all the accessible functionalities are implemented).

Generally speaking,the eight icons correspond to :

◊ the CATALOGUE icon enables the analyst to access to the lists (=catalogues) of the different constitutive elements of an expert system in order to :

- consult a list.
- update a list.
- print a list.
- select an element of a list.

The access to each of the lists is done by selecting one of the following items of the corresponding pop-up :

- EXPERT SYSTEM CATALOGUE.
- KNOWLEDGE BASES CATALOGUE.
- FACTS BASES CATALOGUE.
- SCENARIOS CATALOGUE.

For us these items are particularly relevant because we consider that they correspond to all the components of an expert system. Indeed, the analyst develops an expert system. This expert system formalizes an expert knowledge in one or more Knowledge modules. The expert system will work on one or more Facts bases . To be consulted the expert system needs to be linked to a scenario saying what must be done at a time with the stored knowledge and adapting it to the needs of particular user classes.

◊ the EDITOR icon gives access to a pop-up. There, the analyst can choose an appropriate editor in order to update the following expert system elements :

* for a selected expert system and a selected knowledge module :
  - rules.
  - variables.

* for a selected expert system and a selected facts base :
  - facts.

◊ the REPORT icon gives the opportunity to consult different informations on :

- the knowledge and data the analyst has introduced in a selected expert system.
- the progress of a consultation of a selected expert system.

Let's now present the details about the pop-up items implemented behind this icon :

* *List of rules* :

-> ALL : opportunity to access to all the rules defined in a knowledge module of an expert system (rule names and contents).

-> USING A VARIABLE : to access to a list of rules containing a given variable.

-> USING A VARIABLE IN "IF" CLAUSE : to access to a list of rules containing a given variable in the antecedent part.

-> USING A VARIABLE IN "THEN" CLAUSE : to access to a list of rules containing a given variable in the consequent part.

-> USING UNDEFINED VARIABLES : to access to a list of rules containing variables that have not been defined. This is an important functionality to control the coherence of a knowledge module.

* *List of variables* :

-> ALL : gives the opportunity to access to all the variables defined in a knowledge module of an expert system (variable names and contents).

-> NOT USED : to access to a list of all the variables defined but not used in rules variables. This is an important functionality to control the consistency of a knowledge module.

-> UNDEFINED : to access to a list of all the variables that are used in rules but have never been defined. This is also a functionality useful to control the consistency of a knowledge module.

* *list of facts* :

-> ALL : to access to a list of all the facts of the selected facts base.

-> USING A VARIABLE : to access to a subset of the facts of the selected facts base which contain a given variable.

-> USING UNDEFINED VARIABLES : to access to a subset of the facts of the selected facts base which contain variables which have not been defined.

* *rules network* :

Visualization of a part of the existing rules of the knowledge module of a given expert system through a network.

* *last trace* :

To access to a trace of everything that has happened during the last consultation of the considered expert system.

* *last consultation tree*:

To access to the tree corresponding to the course in the rules network which was performed by the inference engine during the last consultation of the considered expert system.

◊ the EXIT icon gives access to a dialogue box enabling the analyst to conclude his work with K_EXPERT and to return to the DOS system after saving (or not) his session.

◊ the CONSULTATION icon enables the analyst to launch a consultation of a selected expert system from two points of views. Indeed, the corresponding pop-up gives access to :

* *user perspective* :

By selecting this item, the analyst can consult the expert system as if He was a particular user. So during this consultation, He will just have access to the parameters He has affected a user in a given "user presentation".

* *analyst perspective* :

By selecting this item, the analyst can consult the expert system from a test point of view. Indeed, during a consultation of the expert system according to a selected scenario, the analyst can access to different "tools" (e.g. : the contents of the report icon) which enable him to analyze the progress of the consultation in order to improve the expert system quality and efficency.

◊ the CONFIGURATION icon gives access to abilities linked to the fact that the analyst, who works with a computer, must be able to adapt it to his particular needs. The underlying pop-up supports access to the session management (save, restart, delete a session), to the starting of a command language and to the setting of computer parameters such as the choice of a printer and so on.

◊ the COMMUNICATION icon gives access to the ability to copy to another analyst, or from another analyst, some components (expert system, knowledge module, facts base and scenario). This icon is only accessible when an analyst works on a PC connected to a network.

◊ the "ACCESS DOS" icon gives access to the Dos without leaving the K-expert interface.

Presentation:

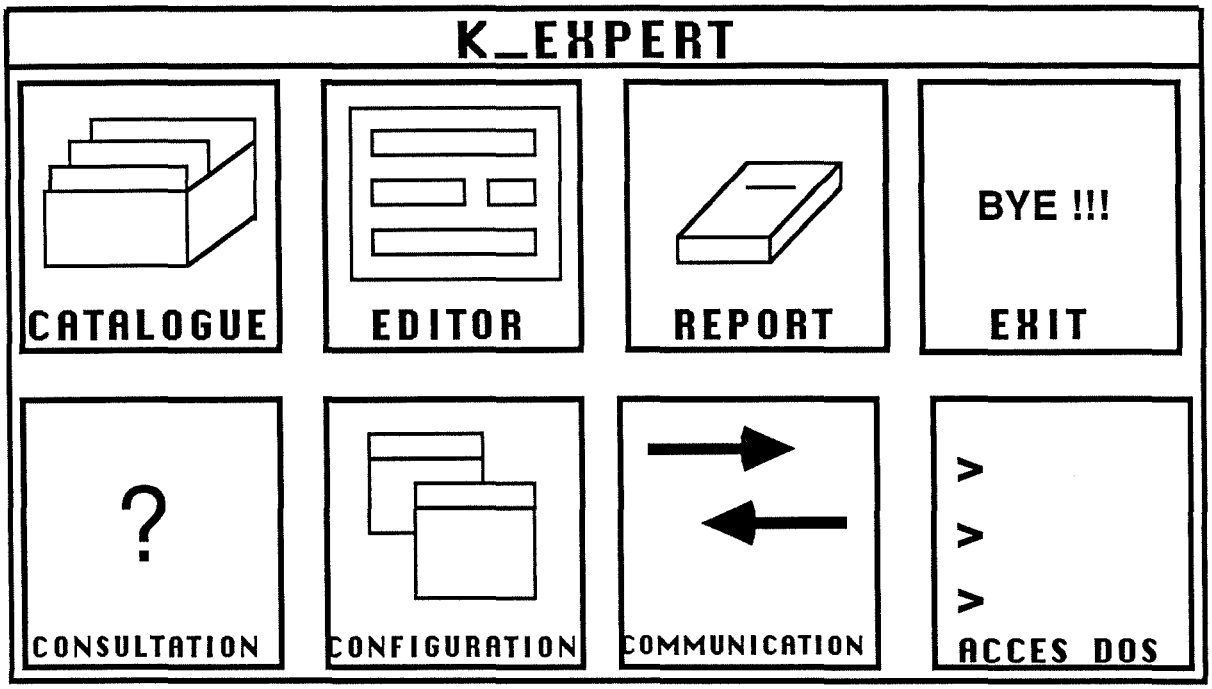


Figure 4.11: K-Expert window.

The associated pop-ups or dialogue boxes are the following ones :

"Catalogue" pop-up

CATALOGUE
Expert Systems
Knowledge Modules Facts bases Scenarios

Figure 4.12: Catalogue pop-up.

"Editor" pop-up

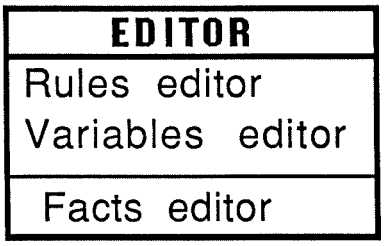


Figure 4.13: Editor pop-up.

"Report" pop-up

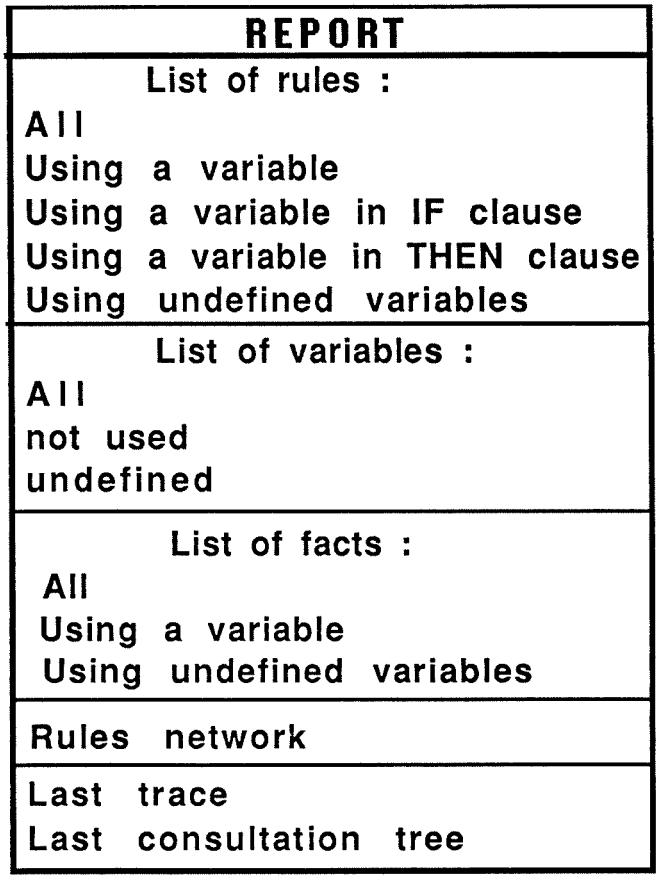


Figure 4.14: Report pop-up.

"Exit" dialogue box

According to the current situation, one of the following dialogue-boxes is displayed.

- if the analyst has not already saved his session.

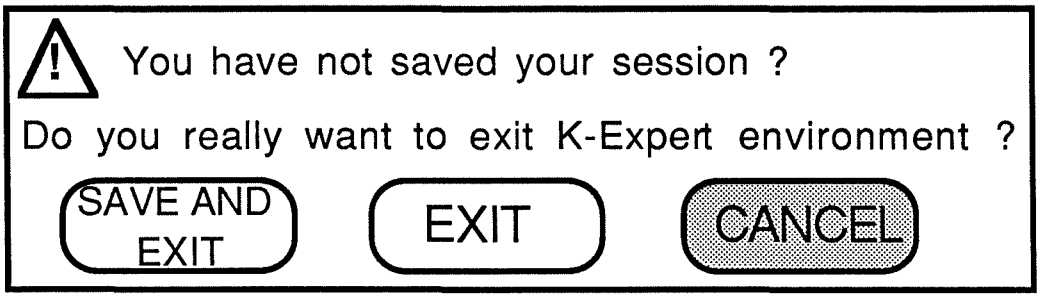


Figure 4.15: Save and exit dialogue box.

- if the analyst has already saved his session.

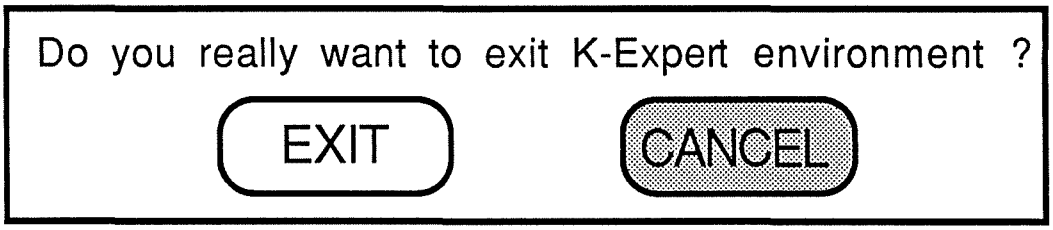


Figure 4.16: Exit dialogue box.

"Consultation" pop-up

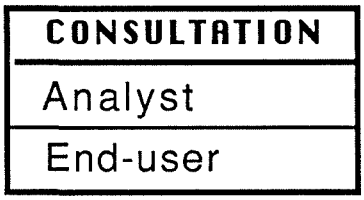


Figure 4.17: Consultation pop-up.

"Configuration" pop-up

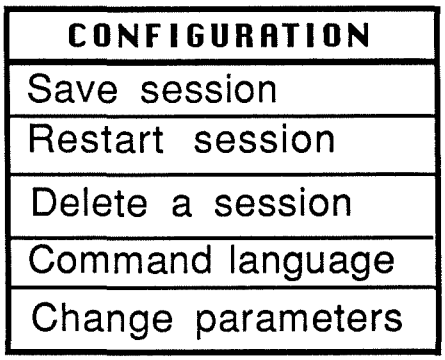


Figure 4.18: Configuration pop-up.

Communication" pop-up

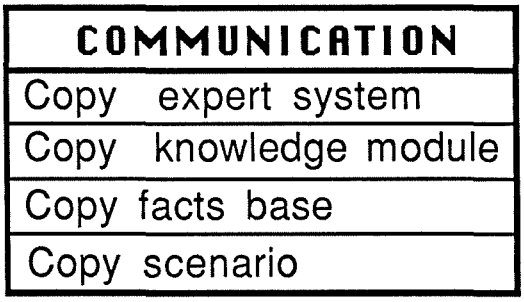


Figure 4.19: Communication pop-up.

"Access dos" window

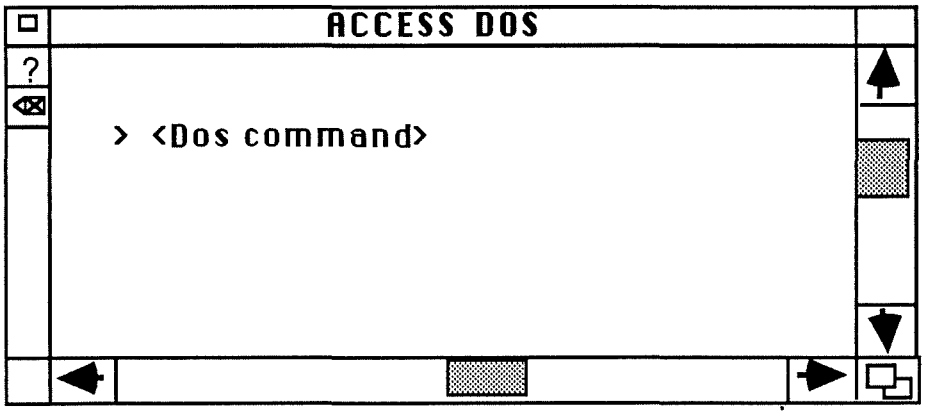


Figure 4.20: Access Dos window.

Actions:

- Window creation: The window is created when the analyst types "K_EXPERT" from the DOS.

- Window closure: mouse-click on the exit icon and mouse-click on the "save and exit" button or on the "exit" button of one of the two possible dialogue boxes of the "exit" icon.

- Affectation of a value to an operation which has to be performed: mouse-click on one of the eight icons. This action triggers off the display of the pop-up corresponding to the category of treatments selected by the analyst. A mouse-click on the desired functionality of the pop-up triggers off the display of the corresponding interactive object.

  To be more precise, we can say that the previously evoked pop-up menus are themselves interactive objects to which some actions are linked. As the way to use and to implement them is really classic, we do not intend to specify each of them in particular.

  What can be done is a general specification of a generic pop-up menu interactive object. Then, it is easy to instantiate it to the particular case of each pop-up.

  Name: <name>-pop-up.

  Definition: menu enabling the analyst to trigger off a particular functionality linked semantically to the others presented in the menu.

Presentation:

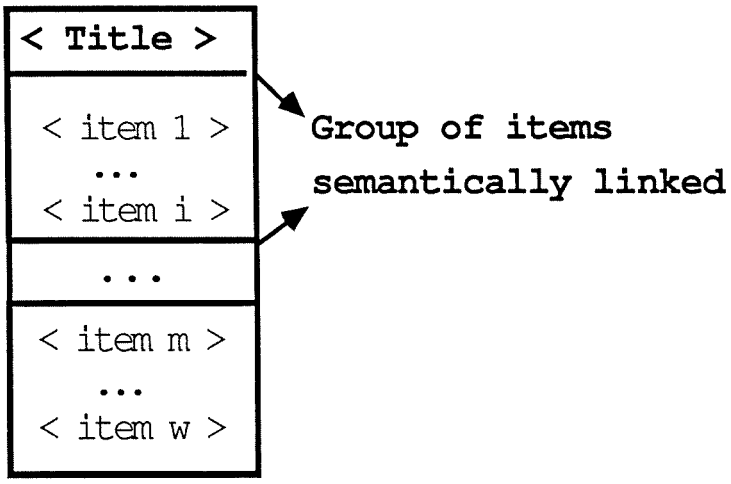


Figure 4.21: Pop-up menu.

Actions:

Menu creation: mouse-click on the corresponding icon.

Menu deletion : mouse release outside one of the menu items.

Affectation of a value to an operation which has to be performed and window closure : mouse-click on the menu item corresponding to the desired functionality. As a result, the pop-up disappears and the interactive object corresponding to the selected functionality is displayed.

In the next pages, a similar description is given for each typical interactive object. For example, we describe only one of the interactive objects corresponding to the "catalogue" or "editor" pop-ups because the functioning principles are similar for each of them. The differences may be found in the objects manipulated (expert systems, knowledge modules...).

4.13.3.2. Interactive objects accessible from the "Catalogue" icon

4.13.3.2.1. Detailed presentation of the "Expert System Catalogue".

Name: Expert-System-Catalogue.

Definition: Window containing all the expert systems already created by the analyst and enabling him to add new ones, to modify or to delete existing experts systems. Moreover, this window is responsible for the display of the current selected expert system by the analyst.

Functionalities: Access to the following functionalities:

- Add an expert system.
- Modify an expert system.
- Delete an expert system.
- Select a current expert system.
- Focus on an expert system.
- Print the name and the description of one or more expert(s) system(s).

Justification: Seeing that all the interactive messages related to the management of expert systems are semantically related, they can be presented through a single interactive object that gives an access to each of them. This should reduce the mental load of the analyst memory and also the number of manipulations required.

Presentation:

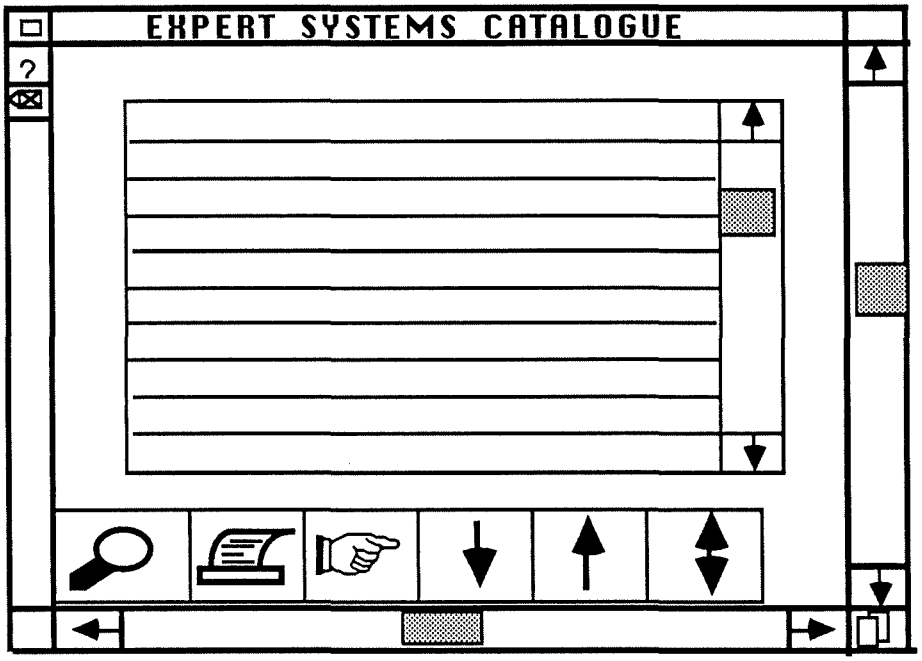


Figure 4.22: Expert systems catalogue.

Actions:

- Window creation: Mouse-click on the corresponding item of the "catalogue" pop-up.

- Window closure: Mouse-click on the close box of the window.

- Affectation of a value to the operation to perform on the expert system displayed : Mouse-selection of an expert system (if necessary) and then, mouse-click on the desired icon of the window to access to the symbolized functionality .

We can now detail the functionalities linked to these ones:

Actions on  : Mouse-click on it to access to the dialogue box enabling the analyst to create a new expert system. This one is an interactive object that may be specified as follows:

Name : Add-expert-system-dialogue-box.

Definition: Dialogue box capturing all the elements necessary to add a new expert system in the list (catalogue) of the existing ones.

Functions: Add an expert system.

Presentation:

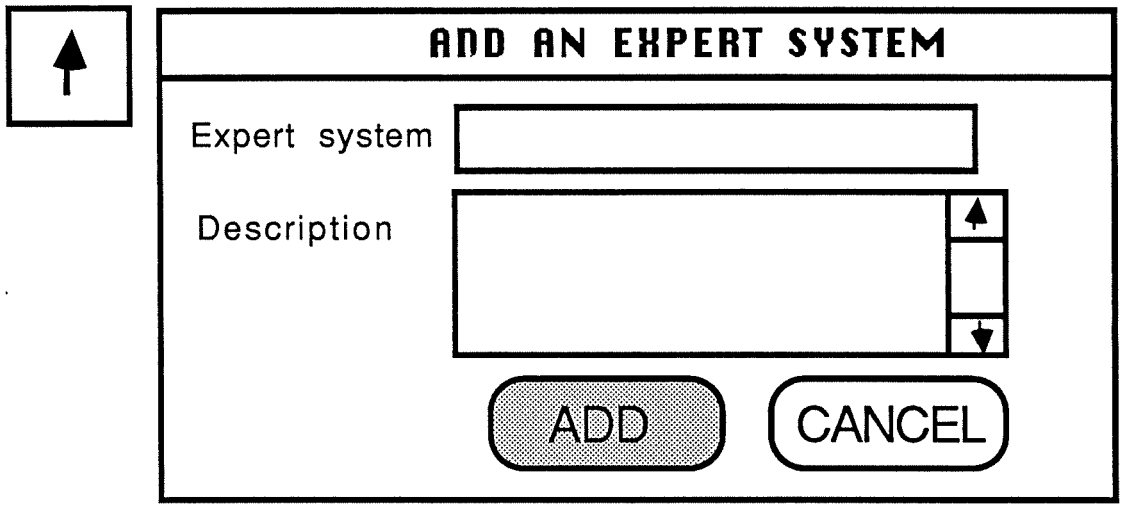


Figure 4.23: Add-an-expert-system-dialogue-box.

Actions:

- Dialogue box creation: mouse-click on the  icon in the "Expert-systems-catalogue " window.

- Dialogue box deletion: mouse-click on the "CANCEL" button. As a result, the analyst is sent back to the "Expert-system-catalogue". The dialogue box disappears from the screen.

- Affectation of a value to a field: Filling of the field with the keyboard.

- Suppression of a field value: Deletion of the field contents with the keyboard.

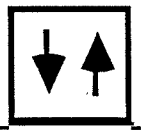
- Correction of a field value: Overwriting of the selected field with a new value.

- Activation of a syntactic control (and if necessary of a semantic one) on a field: Typing on the Return key or switching on the other field by clicking on it with an arrow key after the filling of the field.

  In case of error, an error message (contained in a dialogue box) is displayed to the analyst. This can notably happen if the analyst gives an expert system name that corresponds to an existing one. In this case, a dialogue box (confirmation dialogue box) should be displayed to ask him what He wants to do (overwrite, cancel, ...).

- Dialogue box closure: mouse-click on the "ADD" button or typing on the Return key after a correct filling of the two fields.

Side-effects: After the closure of the "add-expert-system-dialogue-box", the given expert system is created and is added automatically in the catalogue. Moreover, this new catalogue item is highlighted in the catalogue list. The "current-selected-expert-system" field of the "expert-system-catalogue" window is unchanged.

Actions on  : Mouse-click on an expert system displayed in the list of the "expert-system-catalogue" window. Then mouse-click on this icon to access to a dialogue box enabling the analyst to modify the name and description of an existing expert system. As this one may be specified in a very similar way of that used in the case of the dialogue box associated to

the adding of an expert system, we do not go into details. We evoke only its name, its presentation and its side-effects.

Name: Modify-expert-system-dialogue-box.

Presentation:

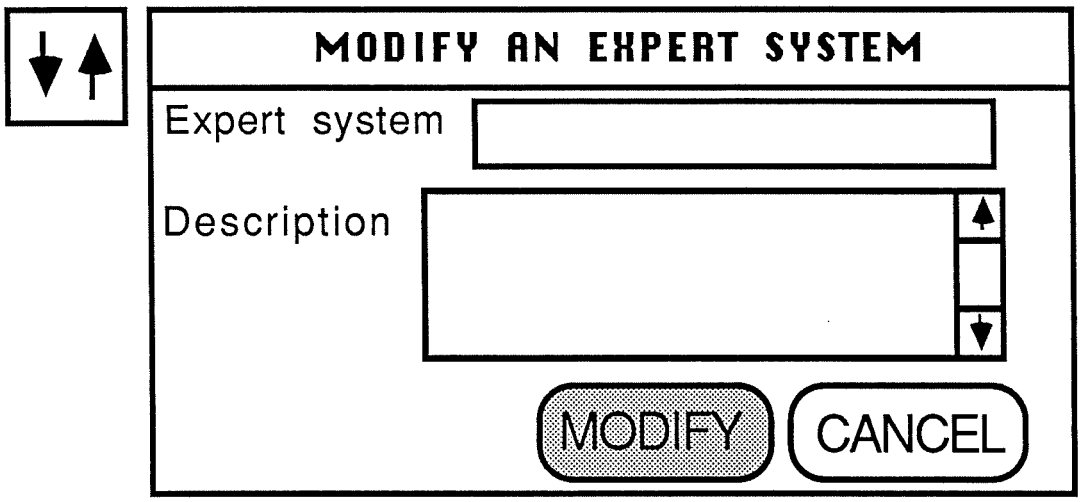
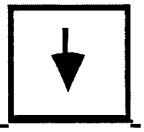


Figure 4.24: Modify-an-expert-system-dialogue-box.

Side-effects: After the closure of the "modify-expert-system-dialogue-box", the given modifications are saved automatically. The highlighted item of the catalogue list is unchanged.

Remark: If the analyst changes the expert system name in such a way that its corresponds to an existing one, an adequate dialogue box containing an error message is displayed to ask the analyst what to do (to overwrite the already existing expert system or to cancel the operation).

Actions on  : Mouse-click on an expert system displayed in the list of the "expert-system-catalogue" window. Then mouse-click on this icon to access to a dialogue box enabling the analyst to delete the selected expert system and all its

components. Finally, mouse-click on one of the dialogue box buttons to confirm the deletion or to cancel it. The dialogue box may be briefly presented as follow.

Name: Delete-expert-system-dialogue-box.

Presentation:

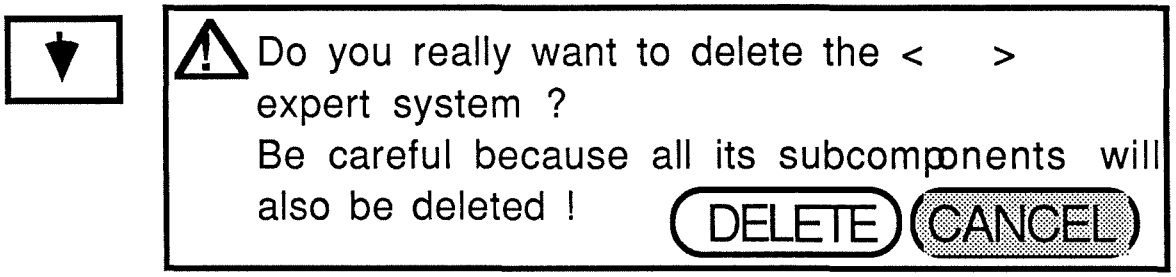
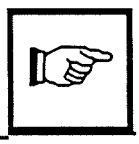


Figure 4.25: Delete-an-expert-system-dialogue-box.

Side-effects: After the closure of the "delete-expert-system-dialogue-box", the given expert system and all its component are automatically deleted from the Data Base. The deleted expert system name is suppressed from the catalogue list. Moreover, the highlighted item of the catalogue list is the one which is just before the deleted expert system or just after it if there is no other expert system name before it in the list. Moreover, if the "current-selected-Expert-System" field of the expert system catalogue window corresponds to the deleted expert system, this field is refreshed.

Actions on  : Mouse-click on an expert system displayed in the list of the "expert-system-catalogue" window. Then mouse-click on this icon to access to a dialogue box enabling the analyst to select the expert system which has been clicked (after a look at its description. About this dialogue box, the following things can be said.

Name: Choose-expert-system-dialogue-box.

Presentation:

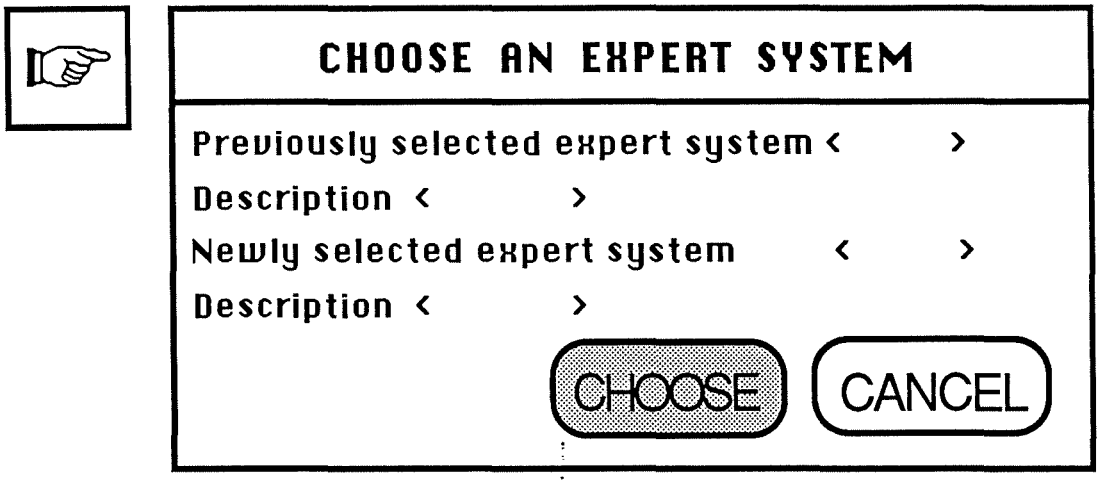


Figure 4.26: Choose-an-expert-system-dialogue-box.

Side-effects: The "current selected expert system" field of the expert systems catalogue window is updated. All the operations related to an expert system that the analyst will perform after this selection are directly linked to the chosen one. The highlighted item of the catalogue list remains unchanged.

Actions on  : Mouse-click on an expert system displayed in the list of the "expert-system-catalogue" window. Then mouse-click on this icon to visualize the description of the selected expert system in a dialogue box characterized by the following elements.

Name: Focus-expert-system-dialogue-box.

Presentation:

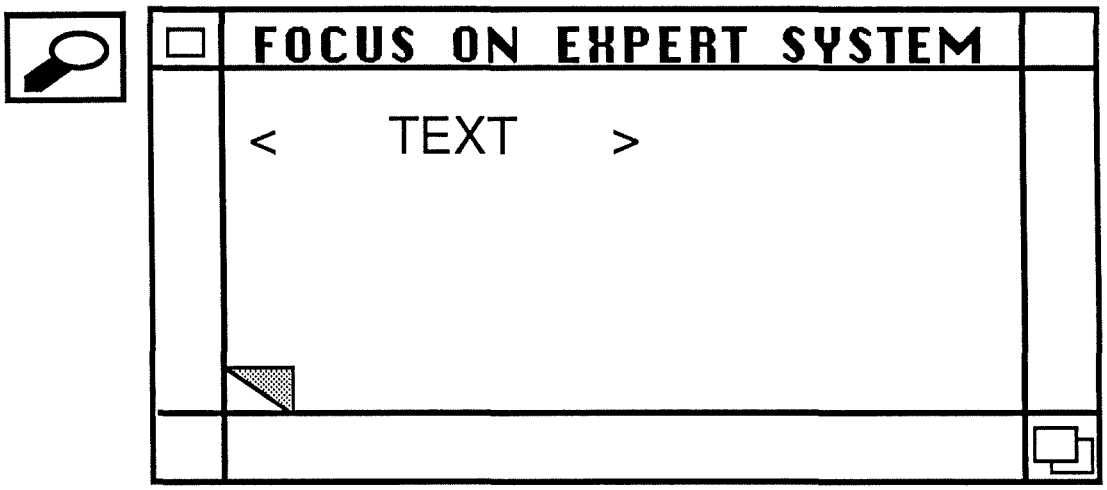


Figure 4.27: Focus-on-expert-system-dialogue-box.

Side-effects: The highlighted item of the expert system catalogue list is unchanged.

Actions on  : Mouse-click on an expert system displayed in the list of the "expert-system-catalogue" window. Then mouse-click on this icon to access to a dialogue box enabling the analyst to print the selected expert system name and its description, or all the expert systems names and their descriptions. The characteristic elements of this dialogue box are:

Name: Print-expert-system-dialogue-box.

Presentation:

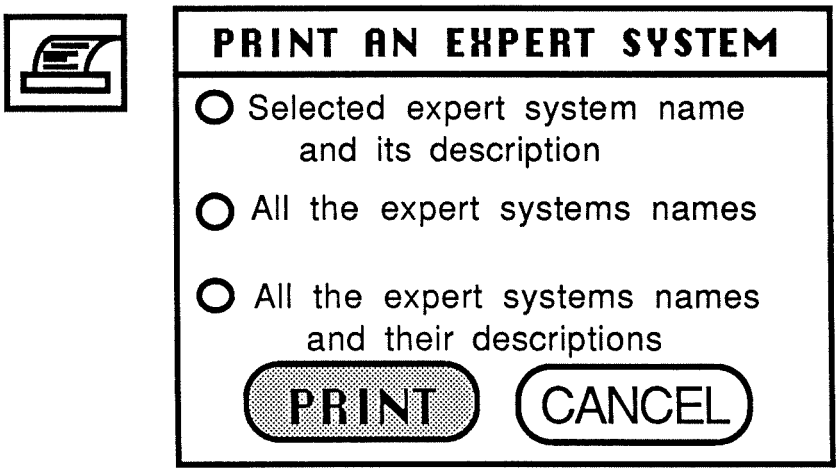


Figure 4.28: Print-an-expert-system-dialogue-box.

Side-effects: If no printer has been selected previously, an error message is displayed in a dialogue box. After acceptation of its contents, the analyst must select a printer (by using the functionalities of the configuration icon)

Now, we evoke briefly the other items of the "Catalogue" pop-up by presenting the interactive objects corresponding to them and by adding some remarks.

This way to do is justified by the fact that the functioning principles of these objects are similar to those applied to the "expert system catalogue" window.

4.13.3.2.2. Knowledge modules catalogue .

Presentation:

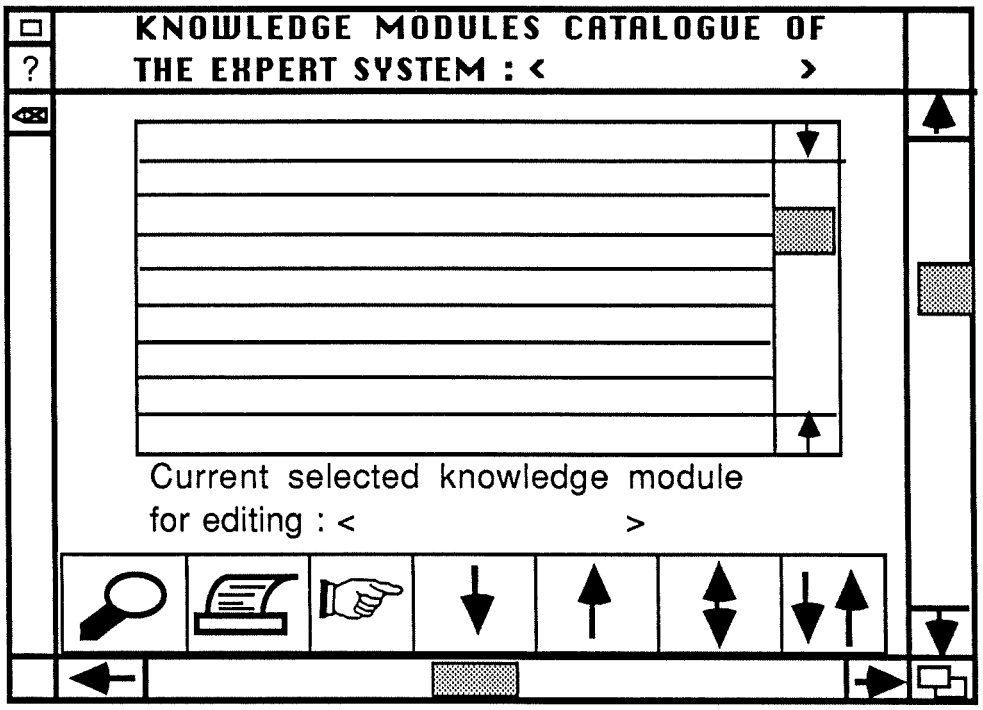
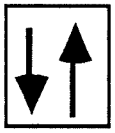


Figure 4.29: Knowledge module catalogue window.

Remarks:

- Before being able to access to this window, the analyst must have chosen a current expert system in the expert system catalogue. If He activates the "Knowledge-modules-catalogue" item without having selected a current expert system first, then a dialogue box containing an error message should be displayed.

- As the icon  has not been evoked in the previous catalogue, we must define it here.

Actions on  : Mouse-click on a knowledge module displayed in the list of the "Knowledge modules catalogue" window that must be copied. Then mouse-click on this icon, to access to a dialogue box enabling the analyst to copy the contents of a given knowledge module in order to reuse the interesting elements in another one. This dialogue box is characterized by the following elements :

Name: Copy-knowledge-module-dialogue-box.

Presentation:

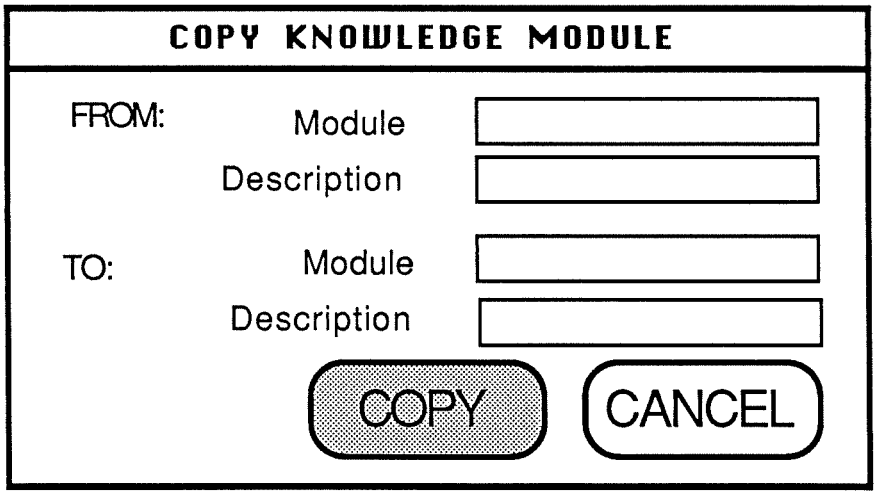


Figure 4.30: Copy-knowledge-module-dialogue-box.

Side-effects: If the targeted knowledge module does already exist, a dialogue box containing an adequate message box is displayed to ask the analyst what must be done (overwrite,merge,cancel). When the new knowledge module is created, its name is automatically added in the catalogue list. The highlighted item of the catalogue list remains unchanged.

4.13.3.2.3. Facts bases catalogue.

Presentation:

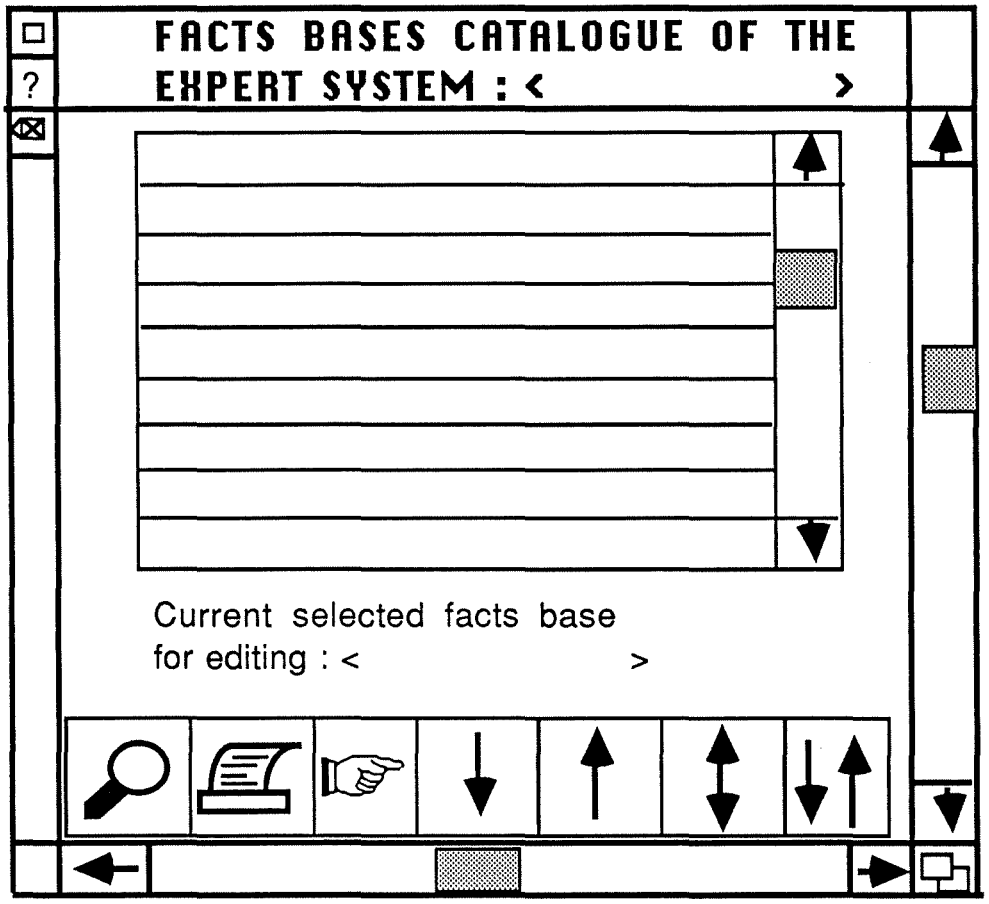


Figure 4.31: Facts bases catalogue window.

Remark: Like in the knowledge modules catalogue an expert system must have been previously chosen. In the other case, a dialogue box containing an error message should be displayed.

4.13.3.2.4. Scenario catalogue.

Presentation:

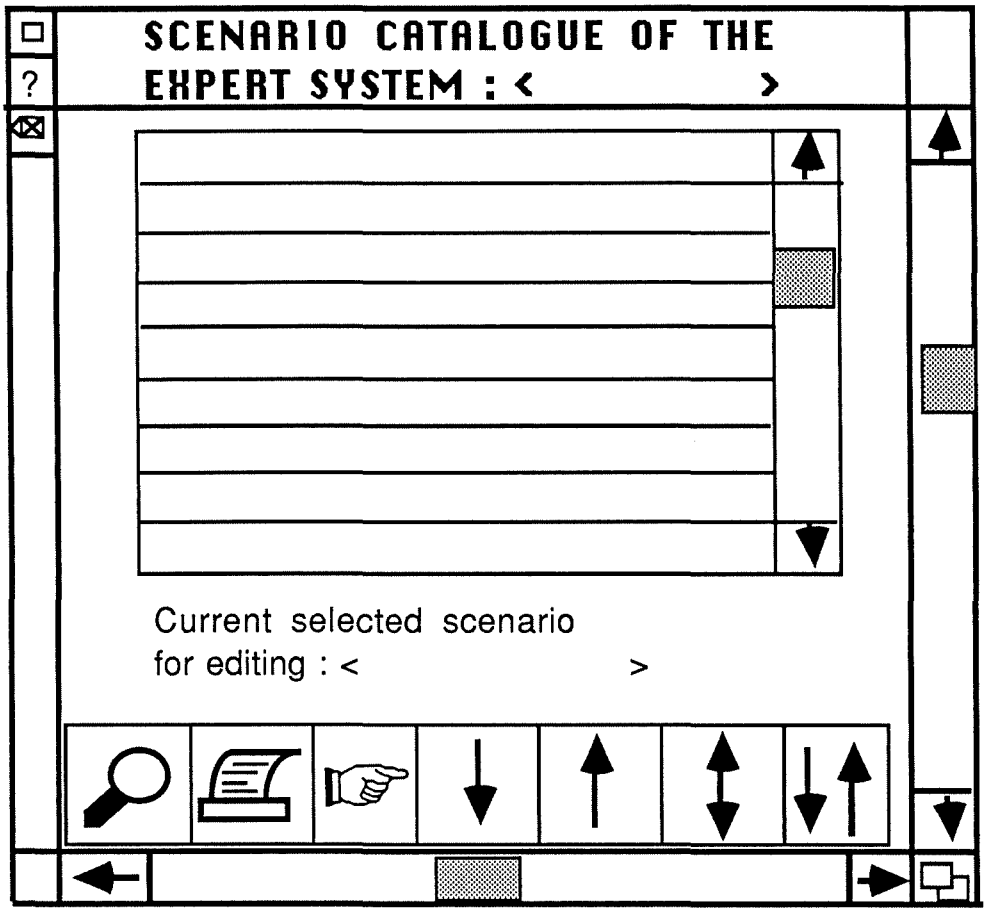


Figure 4.32: Scenario catalogue window.

Remarks: - Like in the two last evoked catalogues , an expert system must have been previously chosen. Otherwise, a dialogue box containing an error message should be displayed.

- Moreover, as the scenarios are directly accessible because they are single elements and are not composed of many sub-elements (like it happens for the knowledge modules and facts bases), a mouse-click on one of the icons contained in the window presented here triggers off directly the display of one of the following dialogue boxes which offer a direct access to access to a scenario contents..

*Add a scenario.

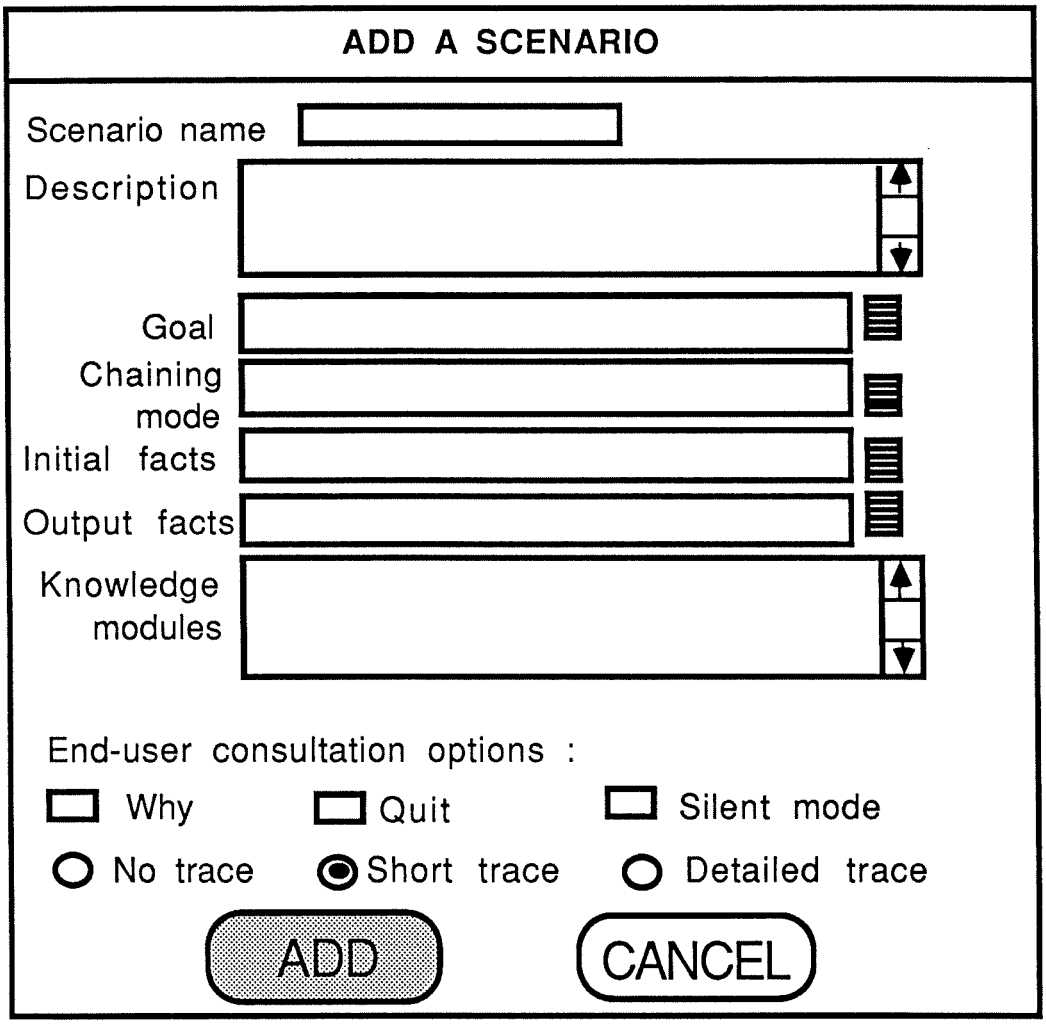


Figure 4.33: Add-a-scenario-dialogue-box.

To be more complete, let's present shortly the contents of the various constitutive fields of such a mask.

Meaning of the fields.

- "Scenario name": identifier of the considered scenario.

- "Description" : explanation of the characteristics of the scenario such as the user class which is concerned.

- "Goal" : goal that must be proved when starting a consultation with this scenario (if there is one).

- "Chaining mode" : chaining mode that must be applied during a consultation with this scenario.

- "Initial facts" : name of the facts base that is furnished to the inference engine when it starts its work with the scenario (if one initial facts base is needed).

- "Output facts" : name of the facts base that the inference engine produces when it obeys to the scenario (if one facts base needs to be produced).

- "Knowledge modules" : name of the modules containing the knowledge on which the inference engine will rely during a consultation based on the scenario.

- "End user consultation options" :

  * "Why" : if selected, during a consultation based on the scenario, the end user will be able to ask the inference engine why it asks him something.

  * "Quit" : if selected, the end-user will be able to quit a consultation at any time.

  * "Silent mode" : if selected, the consultation on the scenario will happen without displaying anything on the screen.

  * "Trace" : Selection of one trace level for a consultation on the scenario.

Similarly to the "Add" scenario, we find also the following options :

Modify a scenario :

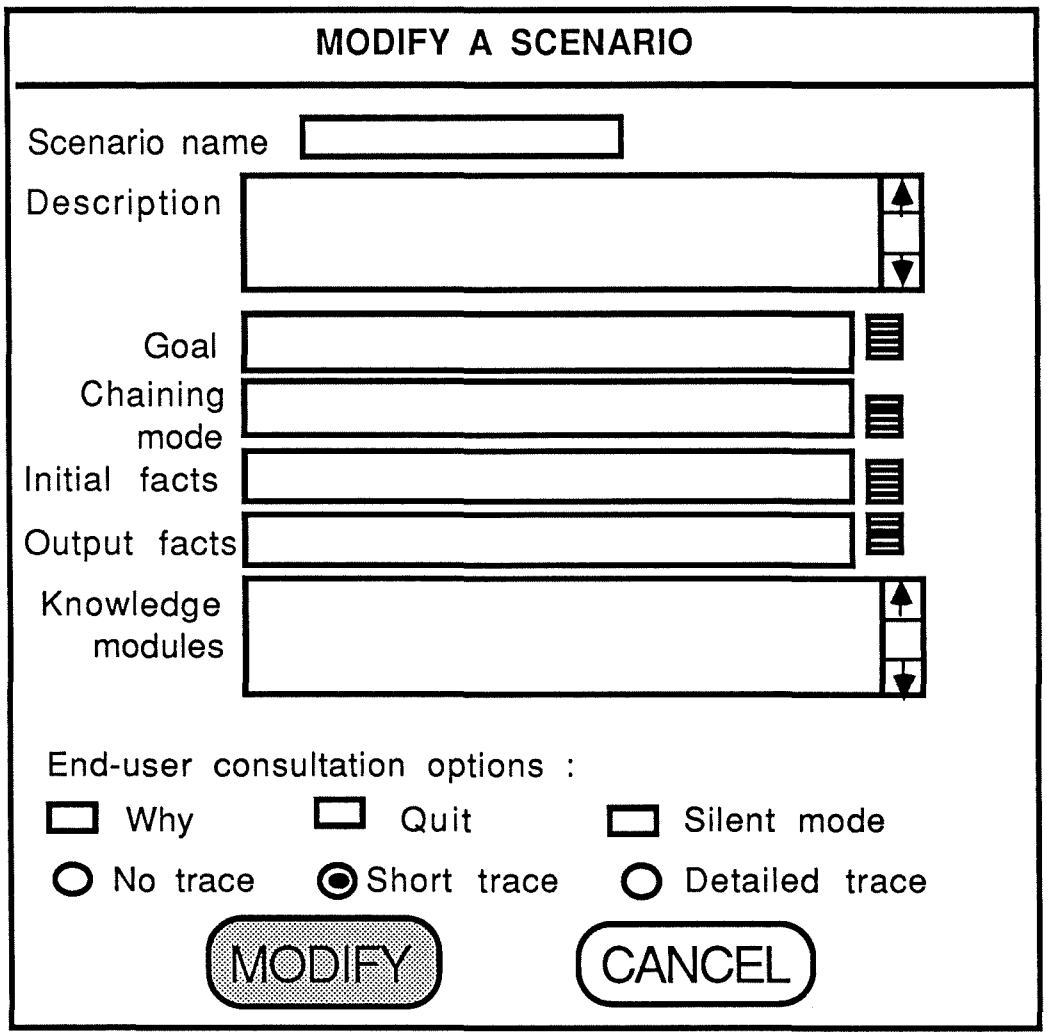


Figure 4.34: Modify-a-scenario-dialogue-box.

Copy a scenario :

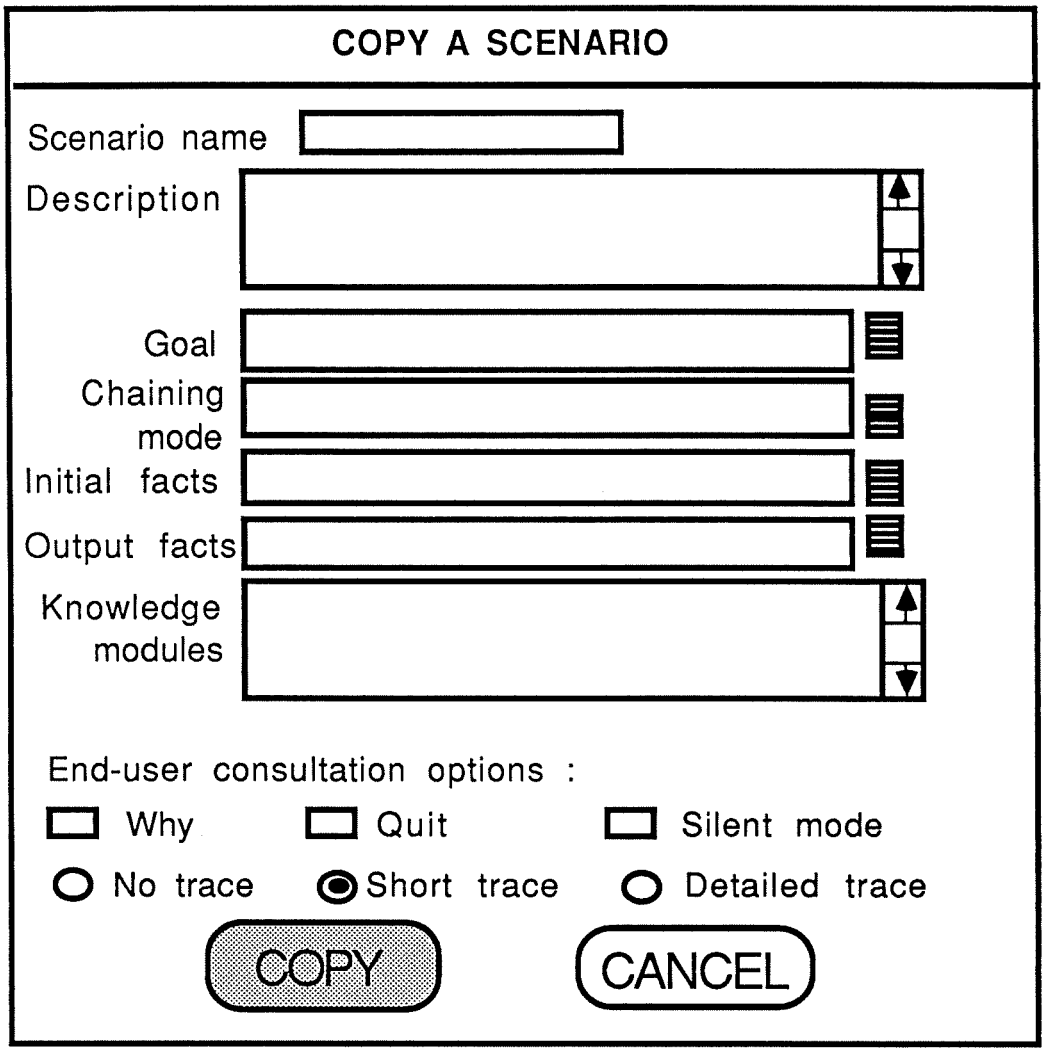
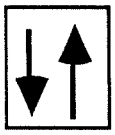


Figure 4.35: Copy-a-scenario-dialogue-box.

Remark: After moving a selected scenario in the catalogue list and after having clicked on the  icon, the analyst receives this dialogue box. Its contents corresponds to that of the selected scenario except that the scenario name field is empty. The analyst may type a new scenario name and modify the contents of all the others fields. Let's note that each analyst's action is protected by dialogue boxes that can appear to signal problems.

4.13.3.3 Interactive objects accessible from the "Editor" icon.

4.13.3.3.1 Detailed presentation of the "Rules Editor".

Name: Rules Editor

Definition: Window presenting a rule mask to the analyst and enabling him to perform all the editing and browsing functions that can be envisaged on such elements.

Functionalities:

- add a rule.
- modify a rule.
- delete a rule.
- print one or more rules.
- copy a rule.
- consult the existing rules (browsing via the flipping mechanism).
- access to a rule of a given name (via the index).

Justification: This interactive object regroups interactive messages related to the management of rules so they are semantical linked and it is not disturbing to present them together to the analyst.

Presentation:

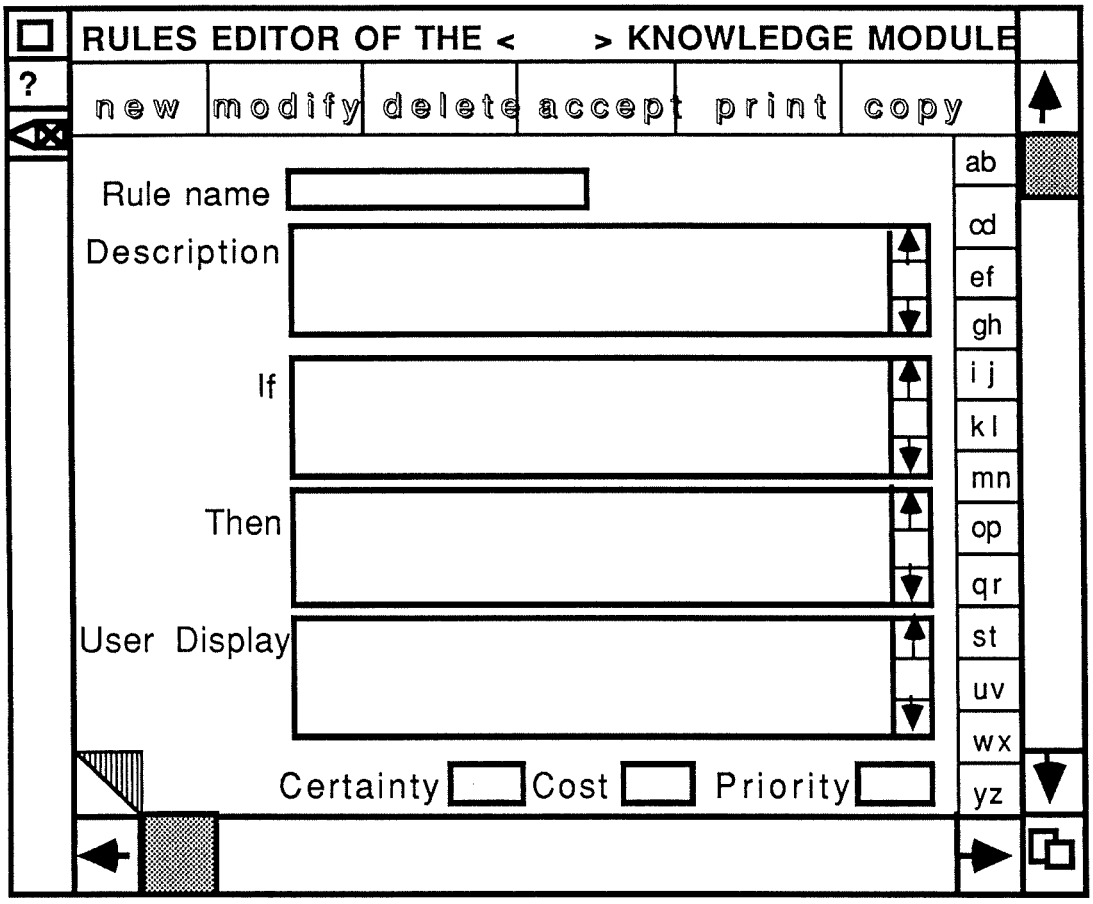


Figure 4.36 : Rules editor window.

Meaning of the fields:

* Rule name : Identifier of a rule.

* Description : Area which enables the analyst to type some comments about the rule.

* If : Antecedent of the rule.

* Then : Consequent of the rule.

* User Display : Information (such as a presentation of the rule in the natural language) that will be displayed in the trace of a consultation to an end-user.

* Certainty : Certainty factor linked to the rule.

* Cost : Cost associated to the rule.

* Priority : degree of priority attached to the rule.

Actions:

Window creation : Mouse-click on the "Rules Editor" pop-up or on the "mask editor" icon of one of the rules list windows presented in Section 4.13.3.4. .

It should be noticed that if the analyst tries to open this window without having selected a knowledge module (and consequently an expert system) first, an error dialogue box should be displayed to signal him this problem. After the acceptation of its contents (mouse-click on the "OK" button of the box), He has to use the catalogue to select one module.

Window closure : Mouse-click on the "close box" of the window

Affectation of a value to a field : Filling of the field with the keyboard.

Deletion of a field value : Deletion of the field contents with the keyboard.

Correction of a field value : Overwriting of the contents of the field with the keyboard.

Activation of a syntactic control (and if needed of a semantic one) : After the filling of a field either typing on the Return key or an arrow key (to move to another field) or click with the mouse on another field. If invalid characters have been introduced, an error message is displayed in a dialogue box.

Affectation of a value to an operation which has to be performed on the displayed rule : Mouse selection of a rule if the desired operation corresponds to another button than NEW. This selection is realized by using the index or the flipping mechanism. Then, mouse-click on the desired button to access the represented functionality. We can now detail the actions linked to these buttons.

* new : Mouse-click on it to display an empty rule mask and to set the cursor into the "Rule name" field in such a way that the analyst is able to fill it directly. During the filling of the mask, this button remains highlighted.

* modify : Mouse-selection of the desired rule then, mouse-click on this button. After this, the cursor is set in the first field of the rule mask and the analyst is able to perform modifications on the contents of all the fields. During the modification of the displayed rule, this button remains highlighted.

* delete : Mouse-selection of the desired rule then, mouse-click on this button to access to a dialogue box asking if the analyst really wants to delete the displayed rule.

* accept : Mouse-click on this button to ACCEPT a new rule or the modifications performed on a displayed rule. It means to save them in the Data Base. This button is only accessible after the performing of a mouse-click on the NEW or MODIFY buttons. In the other case, it appears in grey rather than in dark. When it is clicked on, the previously selected button (NEW or MODIFY) is no more highlighted.

* print : Mouse-selection of the rule to print (if the analyst does not want a global printing) then, mouse-click on this button to access a pop-up enabling the analyst to select what He intents to print.

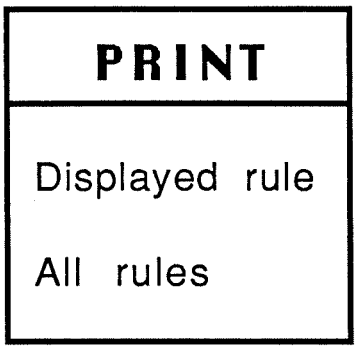


Figure 4.37: Print pop-up.

* copy : Mouse-selection of the rule to copy or of the rule where the copied part must be put. Then mouse-click on this button to set the cursor in the first field of the displayed mask and to display a permanent pop-up enabling the analyst :

  - to copy the whole displayed rule into an album from where it must be retrieved later.

  - to copy one field (completely or partially) of the displayed rule in the album.

    Remark : Before clicking on this item, the analyst has to highlight the field to copy by using the mouse in the mask. The highlighting principle is the following one:

    Move the mouse in the desired field by keeping the left mouse button pressed. If the analyst continues to perform such an operation while switching to another field, the first selected one is no more highlighted because it is only possible to copy one field at a time.

  - to paste the contents of the album into the displayed rule.

    If the album contents corresponds to a complete rule and if the displayed target rule is not an empty one, a previous configuration dialogue box should be displayed.

If the contents of the album is only a field, before clicking on the "paste item", the analyst must locate the cursor at the desired position of the desired field.

- to cancel the displaying of the pop-up because the analyst does not want to perform a copy option.

Remark : after clicking on one of the items of the pop up, this one disappears automatically from the screen.

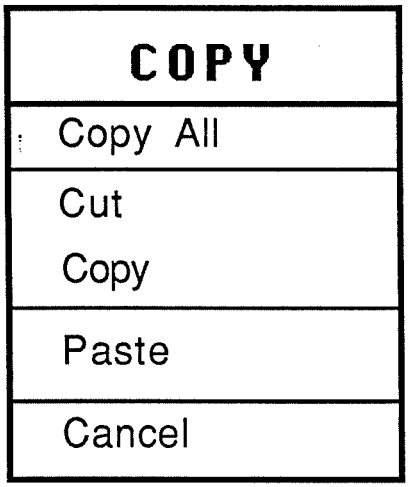


Figure 4.38: Copy pop-up.

We can now present shortly the other editors. Their manipulation principles being similar to those described above, no complementary details are required.

4.13.3.3.2 Variables editor.

Presentation:

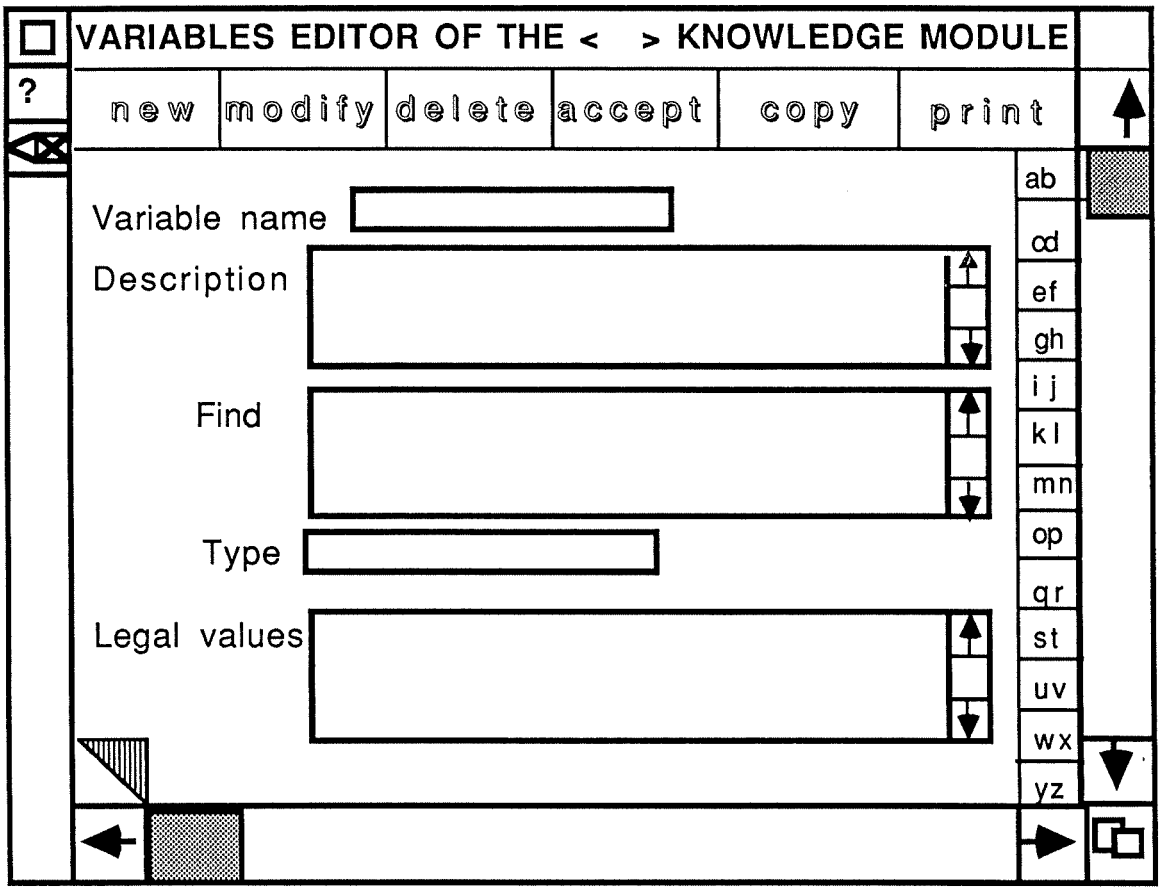


Figure 4.39: Variables editor window.

Fields description :

* "Variable name" : identifier of the variable.

* "Description" : text explaining the meaning and the use of the variable.

* "Find" : Way to find the value of the variable when it is unknown during a consultation. (example : question to the user, triggering off a program...).

* "Type" : Variable type (string, integer...).

* "Legal values" : List of the values that can be attributed to the variable. (For example, a limited range of values for integer type or some pre-defined strings for the string type.)

Remark:

Like in the rules editor, a knowledge module (and consequently an expert system) must have been previously selected via the catalogue. If none is selected, a dialogue box should be displayed to signal that problem to the analyst.

4.13.3.3.3 Facts editor.

Presentation:

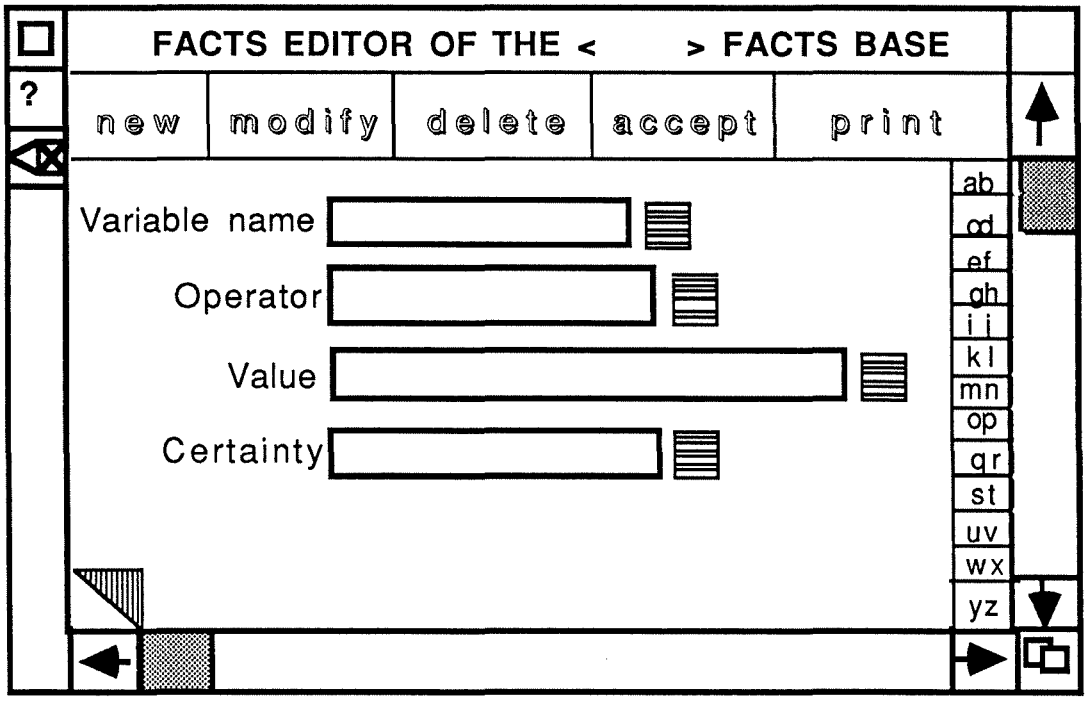


Figure 4.40: Facts editor.

Fields description :

* "Variable name" : Variable identifier.

* "Operator" : Operator linking a variable to a value.

* "Value" : Value concerning the variable.

* "Certainty" : reflection of certainty factor attached by the analyst to the fact.

Remark :

Before being able to access this window, the analyst must have selected a facts base in the catalogue. Else a dialogue box should be displayed to signal that problem.

4.13.3.4 Interactive objects accessible from the "Report" icon.

4.13.3.4.1 Detailed presentation of the "List of rules" window.

Name : List-rules-window.

Definition : window containing a list of all the existing rules belonging to a particular knowledge module of a given expert system. It gives access to the ability to change of expert system and of knowledge module in order to have informations about the rules belonging to various modules. Moreover, it enables the analyst to focus on the contents of one or more rules, to print one or more rules, to access to the rules editor.

Functionalities:- Consult a list of all the rules.

- Print one or more rules.
- Edit rule(s).

Justification: This interactive object regroups interactive messages related to the management of a rules list. So they are semantical linked and it is not disturbing to present them together to the analyst.

Presentation:

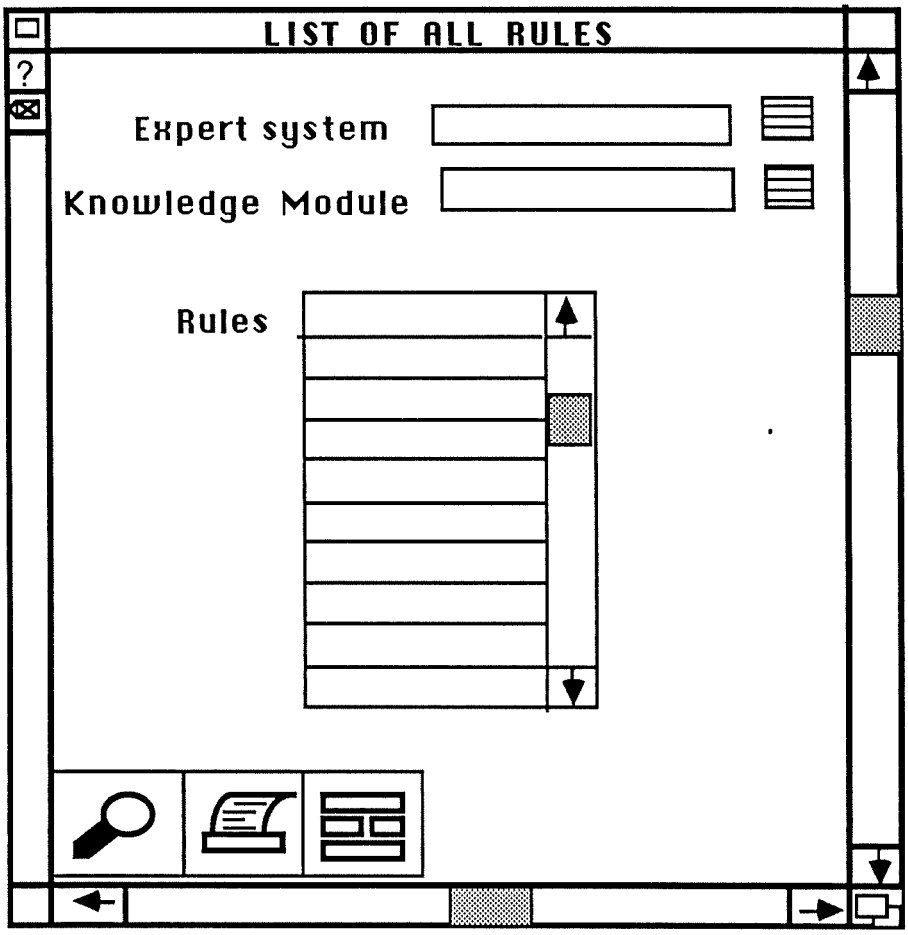


figure 4.41: List of all rules window.

Actions:

Window creation : Mouse-click on the (list of rules) "All" item of the "Report" menu.

Message closure : Mouse-click on the "close box" of the window.

Affectation of a value to a field : Filling of the field with the keyboard of by a mouse-click on the "list" icon.

Deletion of a field value : Deletion of the field with the keyboard.

Correction of a field value : Deletion of the field with the keyboard and then new filling of the field.

Activation of a syntactic control (and if necessary of a semantic one) : Typing on the Return key or on an arrow key (to move to another

field) after the filling of a field. If invalid characters have been introduced or if the given expert system name or knowledge module name are unknown by the system, an error message is displayed in a dialogue box.

Affectation of a value to the operation that must be performed on a displayed rule : Mouse selection of a rule for the desired operation. Then, mouse-click on the chosen icon to access the symbolized functionality.

Now, let's detail the actions attached to these icons :

*  : Mouse-click on the desired rule. Then, mouse-click on this icon to display a window which is similar to the interactive object presented by Figure 4.27 and which make possible the display of the selected rule.

*  : Mouse-click on the desired rule (if the analyst wants to print only a particular rule). Then, mouse-click on this icon to access a dialogue box menu like that presented by Figure 4.28. Of course, this one must be sentenced according to the case of rules.

*  : Mouse-click on the desired rule. Then mouse-click on this icon to access the "rule editor" window. It opens this window or reactivates it if it has already been opened and updates its contents with the selected rule. Moreover, it should be noted that this icon is only accessible if the analyst has not changed the "expert system" and the "Knowledge module" fields. Otherwise, this icon is set in grey. This option should preserve the consistency of the analyst's work by preventing him to switch from one context to another one.

Let's remark also that if this option is activated when a not already accepted rule is displayed in the "rules editor" window which has

been previously opened, a dialogue box should signal this problem to the analyst and ask him how to react. (To accept the already edited rule and to update the editor window, to overwrite the edited rule without accepting the previous one or to cancel the operation.)

Side-effects: Any time, the analyst changes the contents of the "expert system" field the contents of the "list icon" corresponding to the"Knowledge module" field is updated. In the same way, if the analyst changes the contents of the "Knowledge module" field, the "rules list" is refreshed. Moreover, the "editor icon" is "disabled".

In the next pages, we just evoke the presentation of the windows corresponding to the list items of the "Report" pop-up.

4.13.3.4.2 "List of rules using a variable" window

Presentation:

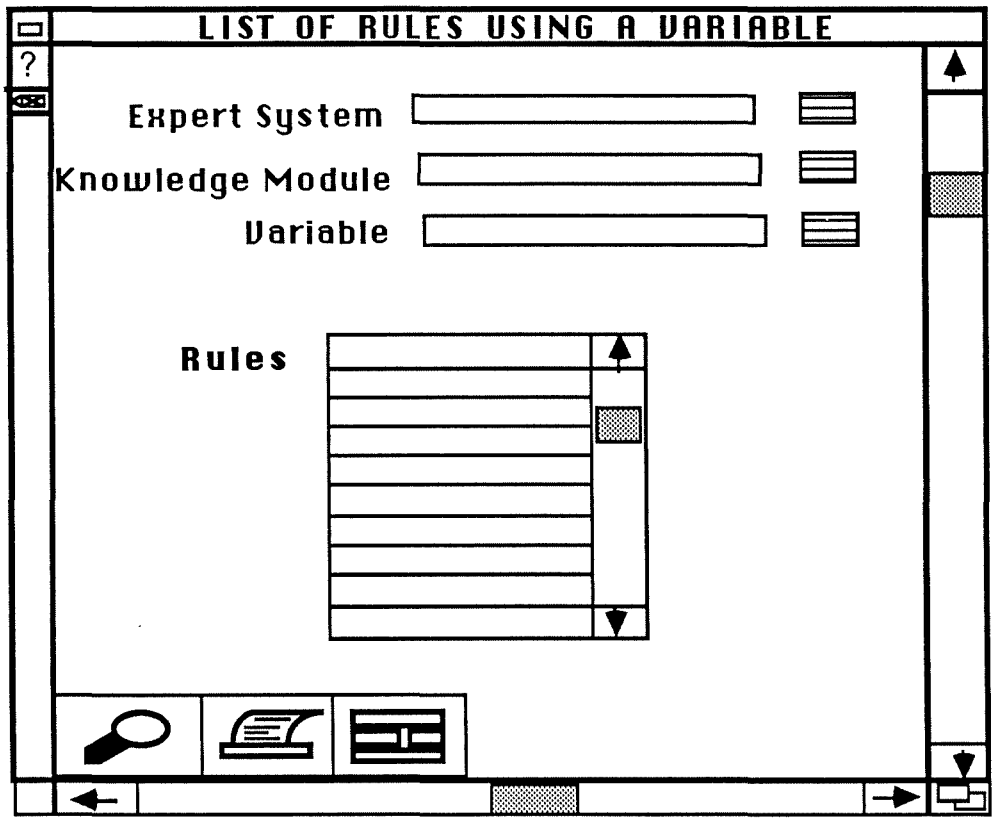


Figure 4.42: List of rules using a variable window.

Remark:

Same principles as those presented for the previous one but this time, a "Variable" field is added. This field enables the analyst to determine which variable may be considered. Let's underline that this variable must correspond to an existing one in the considered knowledge module. Else, an adequate error dialogue box should be displayed.

4.13.3.4.3 "List of rules using a variable in If clause" window.

Presentation:

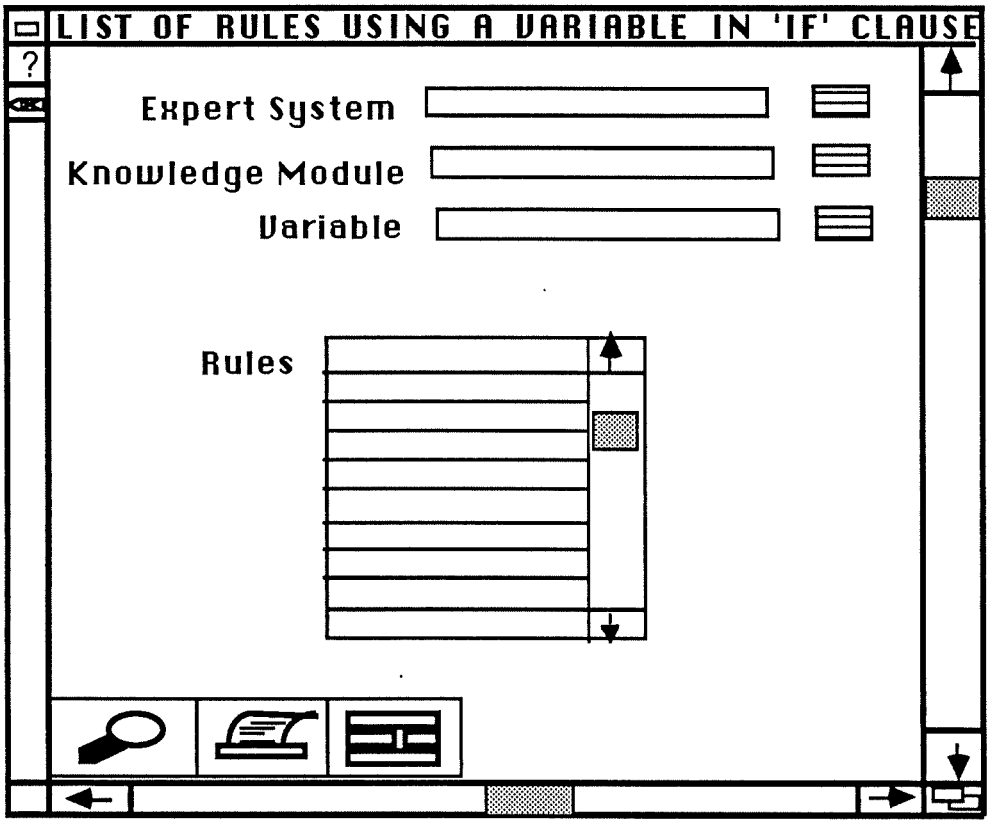


Figure 4.43: List of rules using variable in 'if' clause window.

4.13.3.4.4 "List of rules using a variable in THEN clause" window

Presentation:

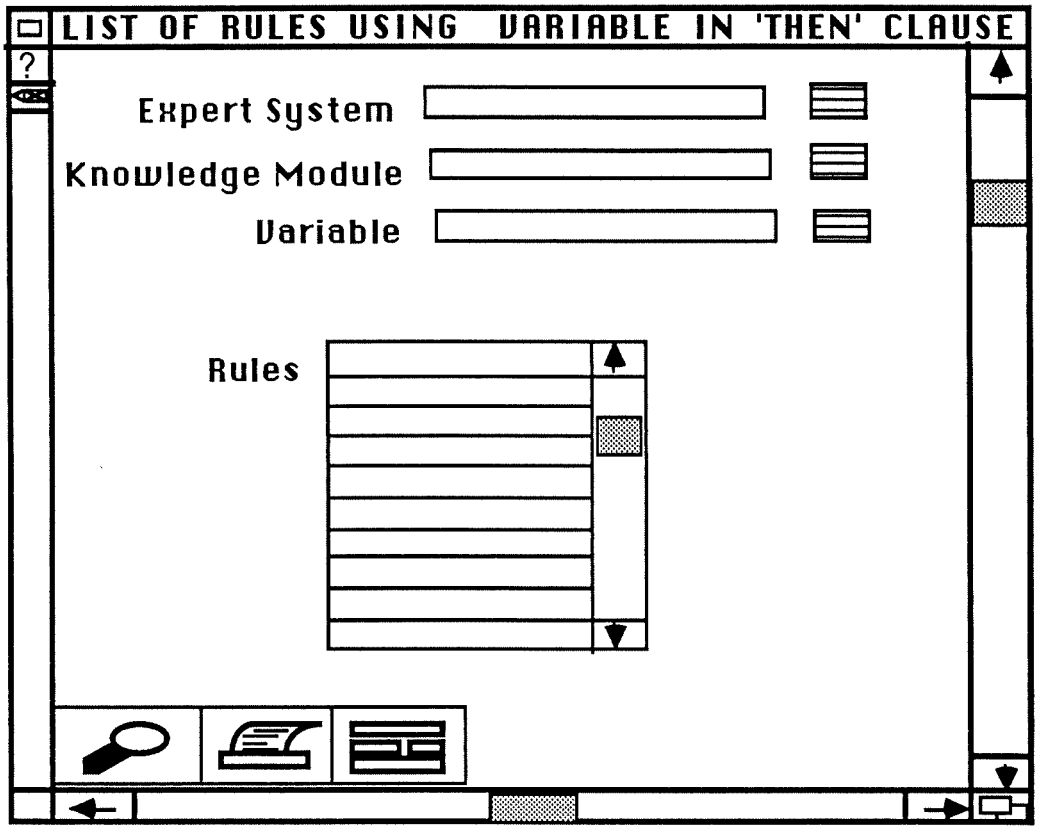


Figure 4.44 : List of rules using variable in 'then' clause window.

4.13.3.4.5 "List of rules using undefined variables" window.

Presentation:

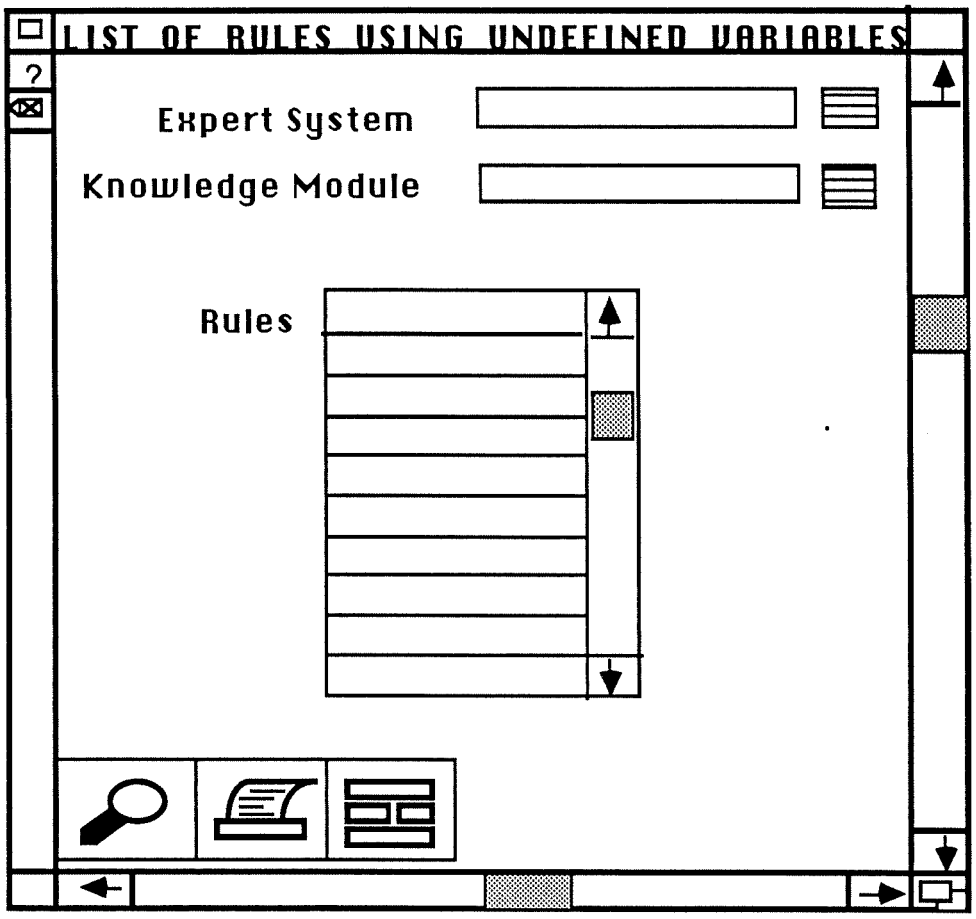


Figure 4.45: List of rules using undefined variables.

4.13.3.4.6 "List of all variables" window.

Presentation:

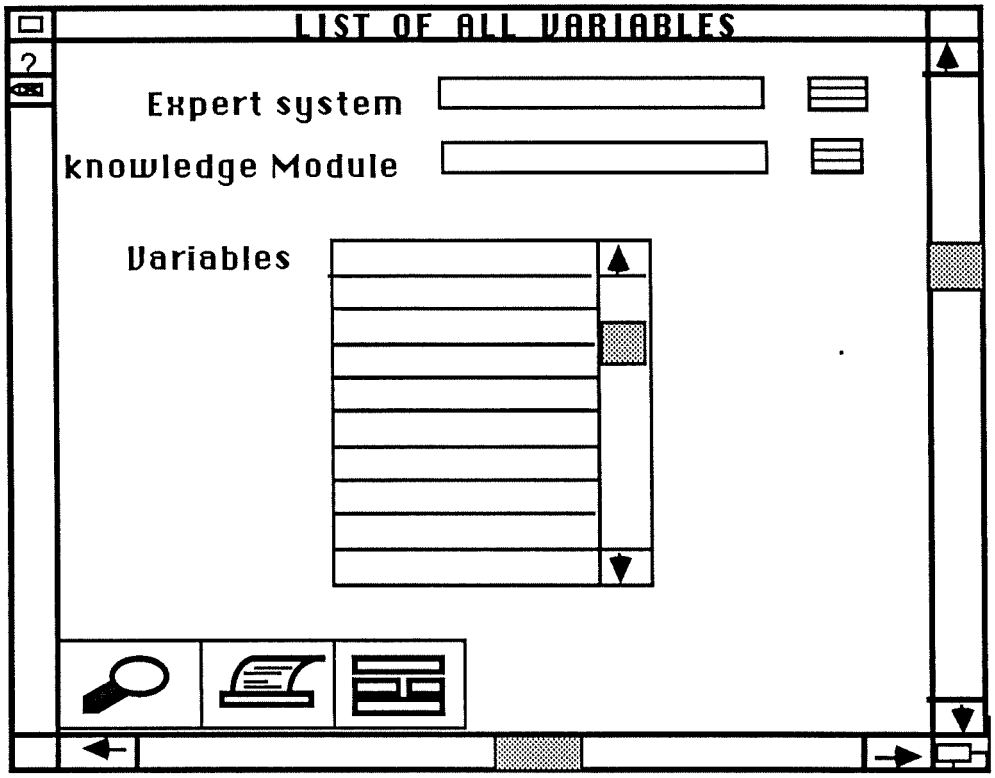


Figure 4.46: List of all variable window.

4.13.3.4.7 "List of not used variables" window.

Presentation:

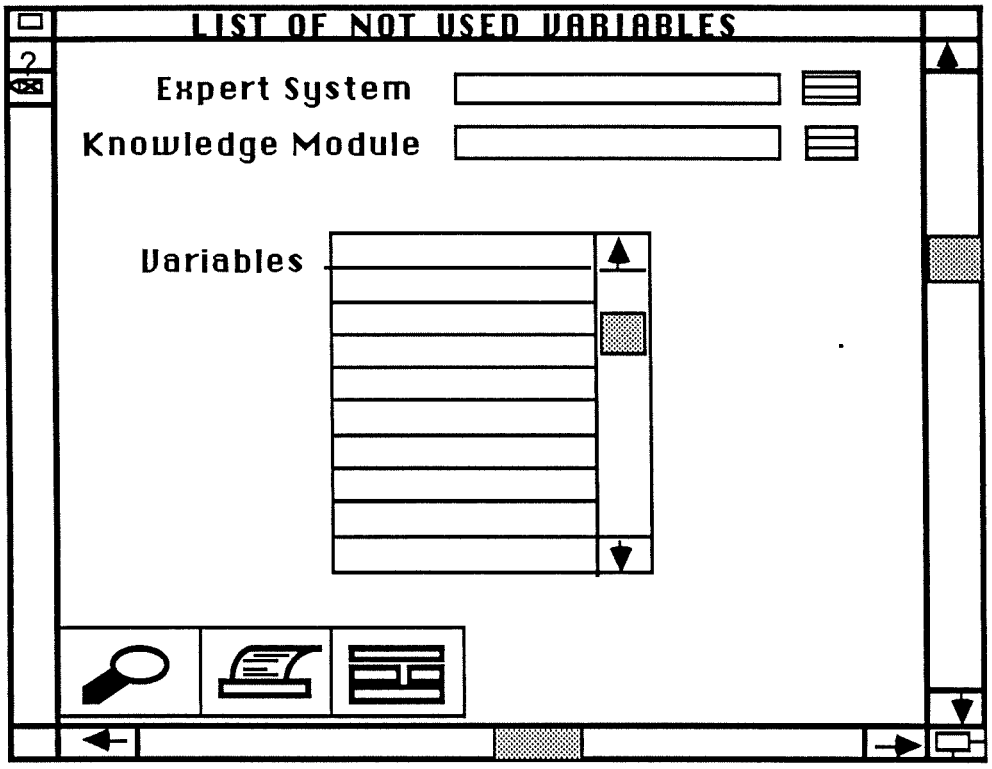


Figure 4.47: List of not used variables window.

4.13.3.4.8 "List of undefined variables" window.

Presentation:

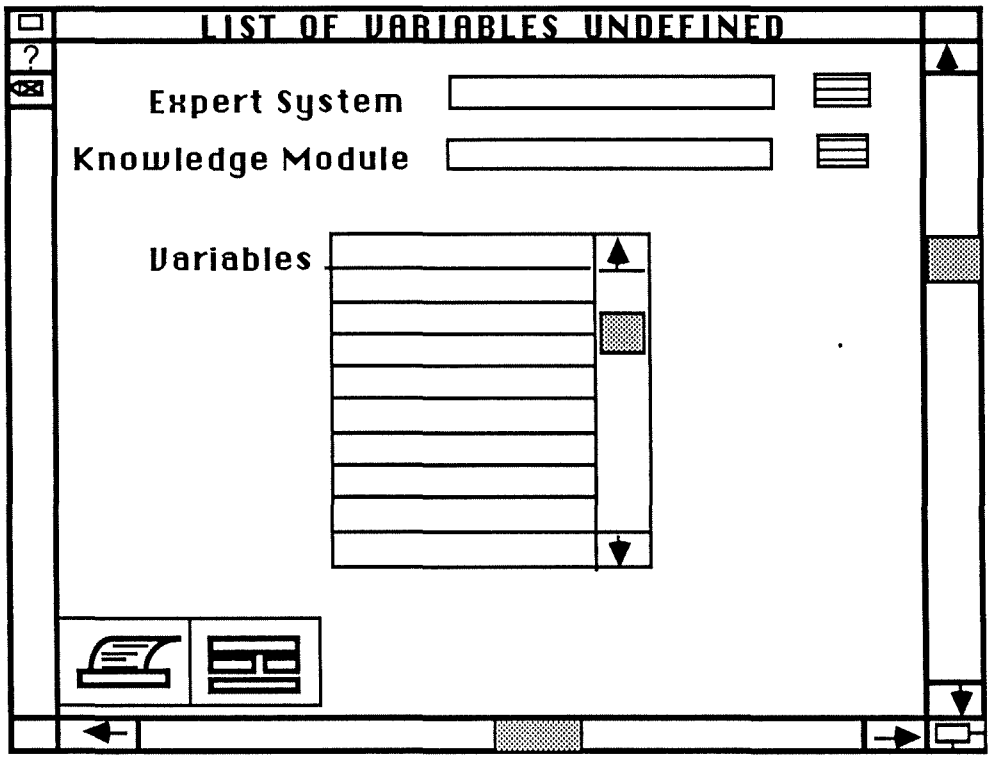


Figure 4.48: List of undefined variables window.

4.13.3.4.9 "List of all facts" window.

Presentation:

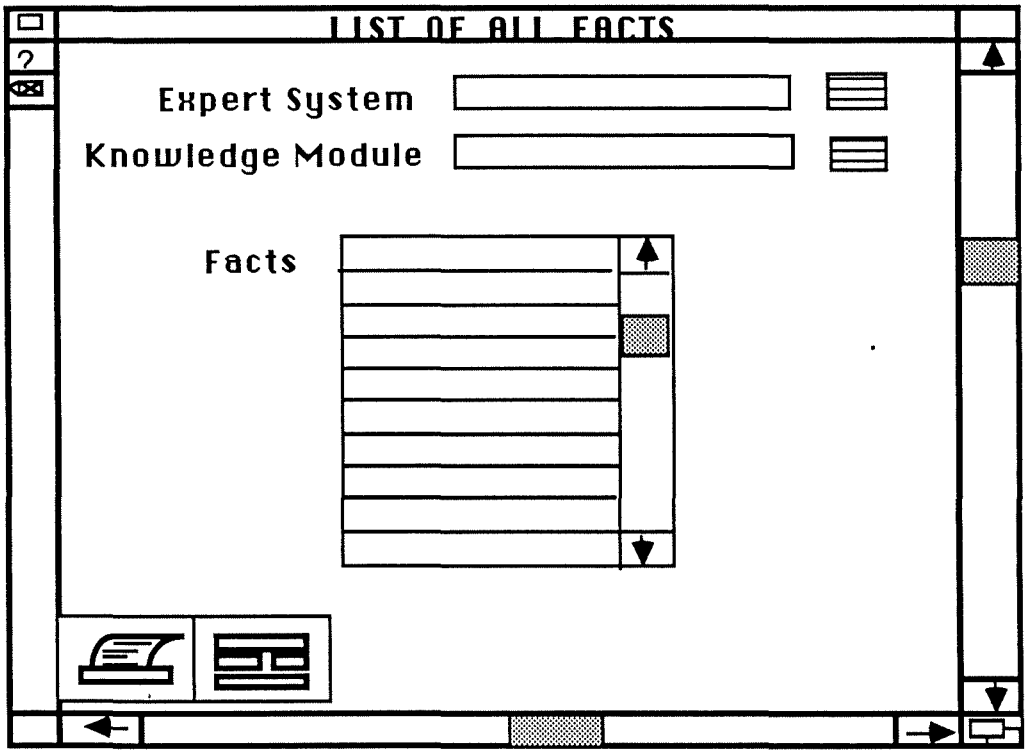


Figure 4.49 : List of all facts window.

4.13.3.4.10 "List of facts using a variable" window.

Presentation:

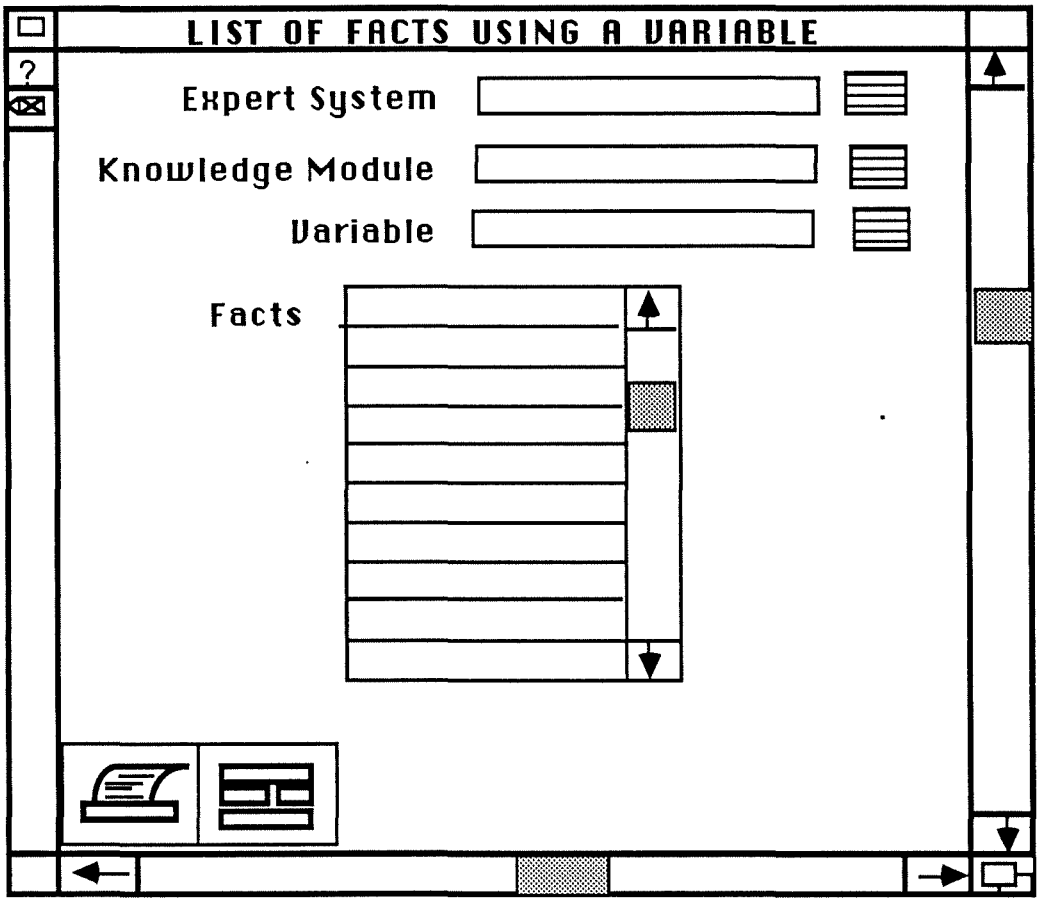


Figure 4.50: List of facts using a variable window.

4.13.3.4.11 "List of facts using undefined variables" window.

Presentation:

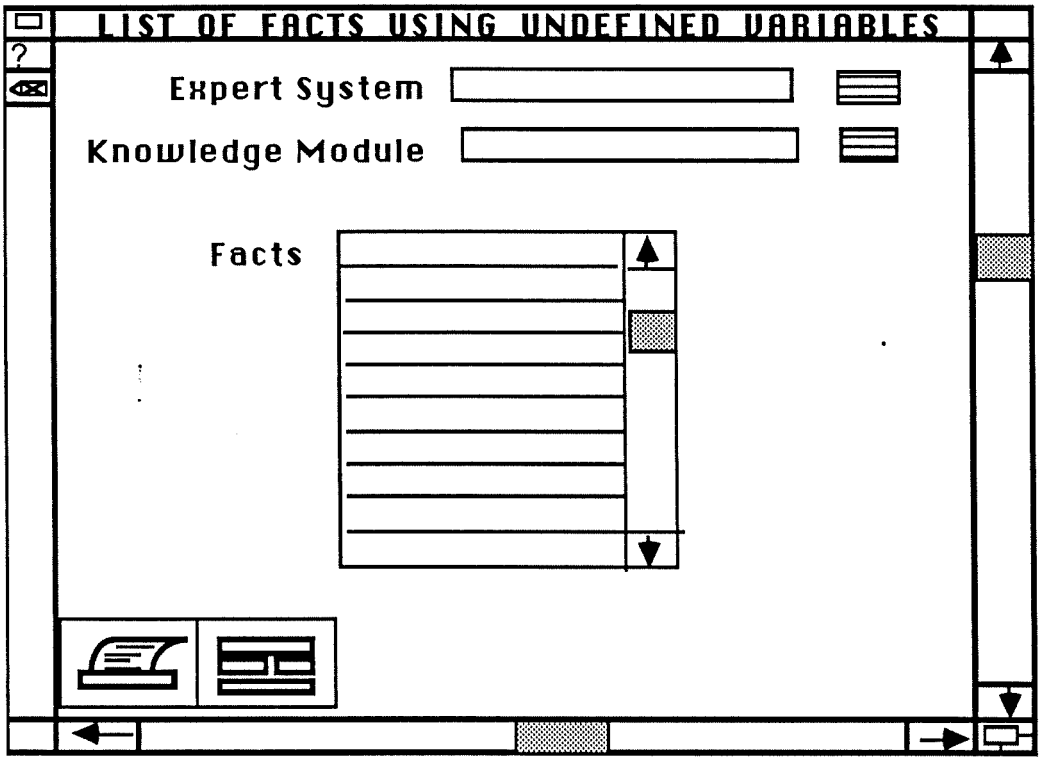


Figure 4.51: List of facts using undefined variables window.

4.13.3.4.12 Detailed presentation of the "Rules network" window.

Name: Rules-network-window.

Definition: Window displaying a part of the whole rules network. This should be a debugging tool. It is only a visualization tool, it does not provide the ability to build graphically a knowledge module. Moreover, in the current state of this work, no graphical formalism for the displaying of rules has been defined.

Functionalities: In fact, the underlying functionality has not been highlighted by the analyst's task analysis. It corresponds to an additional

functionality that can be helpful to support the realization of the retained basic functionalities.

Presentation:

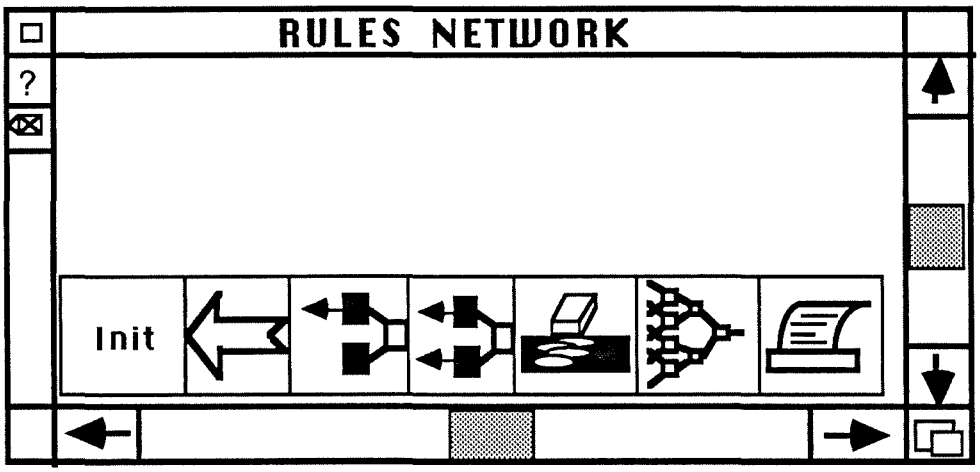


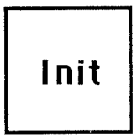
Figure 4.52: Rules network window.

Actions:

Window creation : Mouse-click on the Rules network item of the "Report" icon.

Window closure : Mouse-click on the "Close-box" of the window.

Affectation of a value to the operation to perform on the visualized part of the network : Mouse-click on the desired icon. We are going to describe the actions attached to each icon.

* 

Mouse-click on it to access a pop-up enabling the analyst to start the display of the network from a chosen variable or from a chosen rule or to display the whole network.

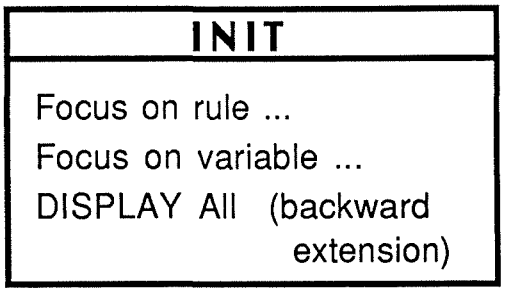


Figure 4.53: Init pop-up

Let's remark also that a mouse-click on one of the two first items of this pop-up triggers off the display of one of the two following dialogue boxes. By typing a value or selecting one for the rule or variable field, the analyst has the ability precise from where He wants to display the network.

Focus-on-rule dialogue box :

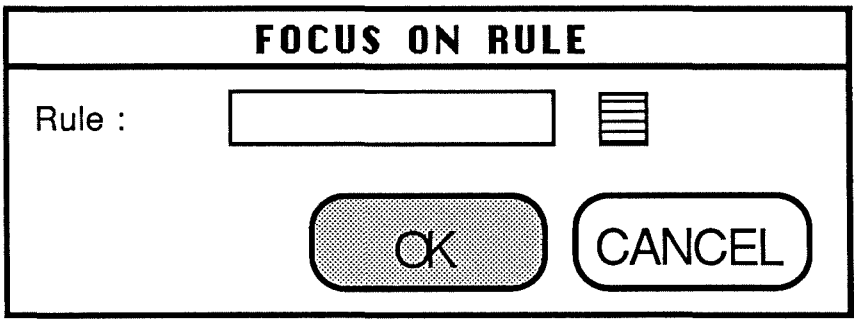


Figure 4.54: Focus-on-rule-dialogue-box.

Focus-on-variable dialogue box :

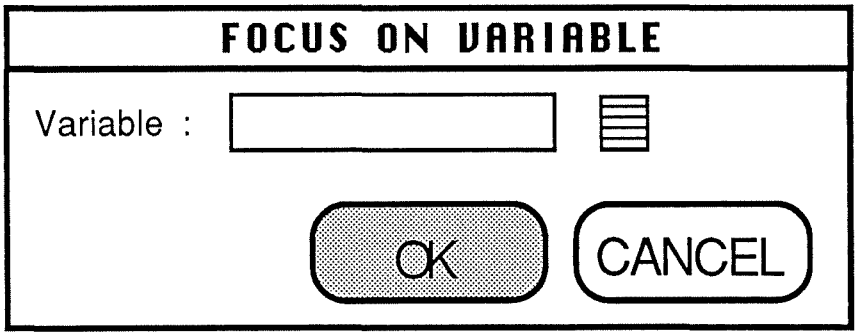
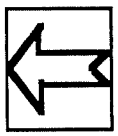


Figure 4.55: Focus-on-variable-dialogue-box.

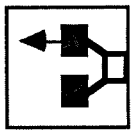
*  Mouse-click on this icon. Then, mouse-click on :

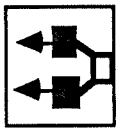
- item of an "If" clause of a rule in the network in order to display a left extension containing all the rules in which the item appears in the "Then" clause.

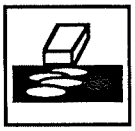
- an item of the "Then" clause of a rule in the network to display a right extension containing all the rules in which the item appear in the "If" clause.

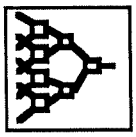
- the left side of the focused variable to display a left extension containing all the rules in which the variable appears in the "Then" clause.

- the right side of the focused variable to display a right extension containing all the rules in which the variable appears in the "If" clause.

*  Mouse-click on this icon. Then mouse-click on one of the four elements presented for the previous icon. However, this time, the extension (left or right) is built as far as possible by recursivity and not limited to one step. If one element has already been developed in another part of the tree, this one is not extended again.

*  Same principles as those attached to the previous icon but this time, all the elements, without any exception, are developed.

*  Mouse-click on this icon. Then, mouse-click on an element of the network that must be deleted. This provokes also the deletion of dependent elements which are no longer related to other parts of the network.

*  Mouse-click on this icon to open a window containing an overview of the whole network and enabling the analyst to move the network area visualized through the "Rules network" window. This could be implemented by a way similar to that used in the Nexpert Network Overview which has been evoked in section 3.3.

*  Mouse-click on this icon to access a dialogue box similar to that presented by figure 4.2.3 and linked to the same side-effects.

Side effects: Whenever one of the "development" icon or the "eraser" icon is clicked on, this one becomes highlighted up to the analyst decides to click on another one or to access to another window. By this way, the analyst may perform the same action on different elements of the network consecutively without having to click on an icon several times.

4.13.3.4.13 "Last trace" window.

Name: Last-trace-window.

Definition: Window containing the text of the trace which has been built in parallel to the last consultation of an expert system selected in the catalogue.

Functionalities: In fact, the underlying functionality has not been highlighted by the analyst's task analysis. It corresponds to a debugging functionality

that can be helpful for the analyst. In fact, out of a consultation, the analyst has the ability to display the trace of the last consultation He has performed.

Presentation:

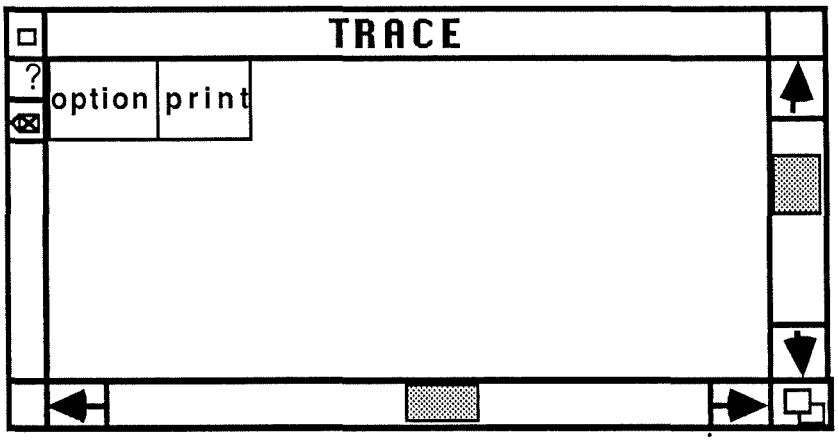


Figure 4.56: Last trace window.

Actions:

Window creation : Mouse-click on the "last trace" item of the "Report" pop-up.

Window closure : Mouse-click on the "Close box" of the window.

Affectation of a value to the operation to perform on the trace : Click on the desired button.

* print : Mouse-click on the "print" button to access to a dialogue box asking if the analyst really wants to print the trace. If no printer has been selected a dialogue box containing an error message should be displayed.

* option : Mouse-click on this button to access a pop-up enabling the analyst to choose the trace level.

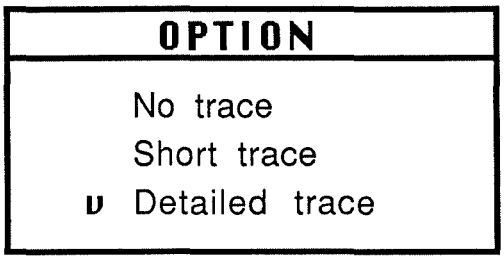


Figure 4.57: Option pop-up

4.13.3.4.14 "Last consultation tree" window.

Name: Last-consultation-tree-window.

Definition: Window containing the picture of the rules tree built during the last consultation on the expert system selected in the catalogue.

Functionalities: In fact, the underlying functionality has not been highlighted by the analyst's task analysis. It corresponds to a debugging functionality that can be helpful for the analyst. In fact, out of a consultation, the analyst has the ability to display a tree showing the reasoning performed during the last consultation.

Presentation:

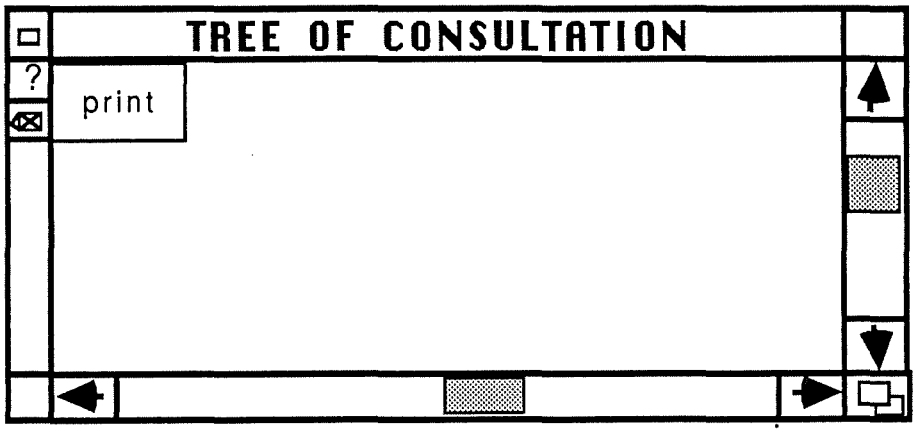


Figure 4.58 : Last consultation tree window.

Actions:

Window creation : Mouse-click on the last consultation tree of the "Report" pop-up.

Window deletion : Mouse-click on the "Close box" of the window.

Affectation of a value to the operation to perform on the tree : Mouse-click on the "print" button to access to a dialogue box asking if the analyst really wants to print the tree. If no printer has been selected, a dialogue box containing an error message should be displayed.

4.13.3.5 Interactive objects accessible from the "Consultation" icon.

4.13.3.5.1 Detailed presentation of the "Analyst-consultation" window.

Name: Analyst-consultation-window.

Definition: Window containing the questions to be answered during a consultation and giving access to consultation debugging options. At the end of a consultation, the question field is replaced by a text providing the result of the consultation.

Functionalities: Consult an expert system (from an analyst's point of view).

Justification: This interactive objects regroups interactive messages related to the consultation of an expert system in order to debug it.

Presentation:

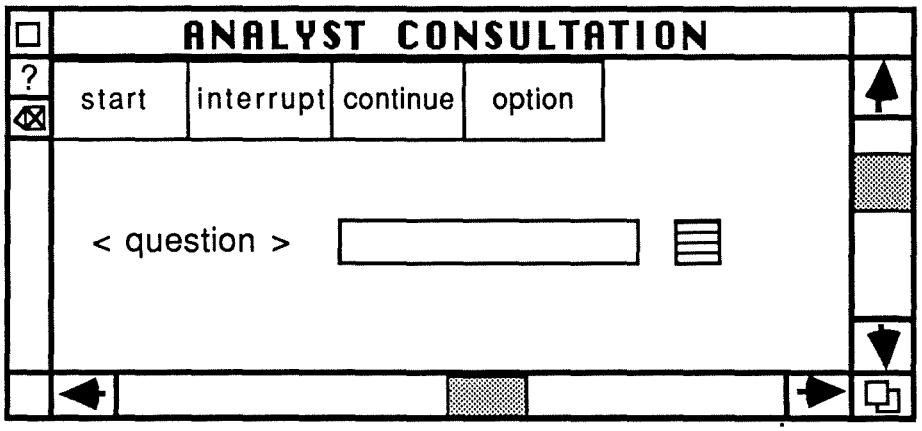


Figure 4.53: Analyst consultation window.

Actions:

Window-creation: Mouse-click on the (consultation) "analyst" item of the "consultation" pop-up. If no scenario has been selected previously in the catalogue, a dialogue box containing an error message should be displayed to signal the problem to the analyst .

Window closure: Mouse-click on the "close-box" of the window.

Affectation of a value to the question field: Filling of the field with the keyboard or by a mouse-click on the "list" icon.

Deletion of the question field value : Deletion of the field contents with the keyboard.

Correction of the question field value: Filling of the field with a new value (with the keyboard of by a mouse-click on the "list" icon).

Activation of a syntactic control and semantic control on the question field : Typing on the "return" key after the filling of the field. In case of error, an error message (contained in a dialogue box) is displayed to the analyst.

Affectation of a value to the operation to perform at a certain step of the consultation : Mouse-click on the desired button. Let's detail the actions attached to each button.

* start : Mouse-click on this button to start a consultation of the selected expert system on the selected scenario.

  Side effect : When this button is clicked, it is highlighted. It stays in this state up to a next mouse-click or up to the end of the consultation.

* interrupt : Mouse-click on this button to interrupt a consultation of the selected expert system on the selected scenario.

Side effect : When this button is clicked, it is highlighted. It stays in this state up to the analyst clicks on the "continue" button. We can also note that this button is enabled (e.g. : in grey color) only if the analyst has started a consultation.

* continue : Mouse-click on this button to continue an interrupted consultation.

  Side-effect : When this button is clicked on, it is highlighted. When the consultation is started again, the "interrupt" button is no more highlighted. This button is enabled only if an analyst has interrupted a consultation.

* option : Mouse-click on this button to have access to a pop-up enabling the analyst to activate various debugging options that are summarized now.

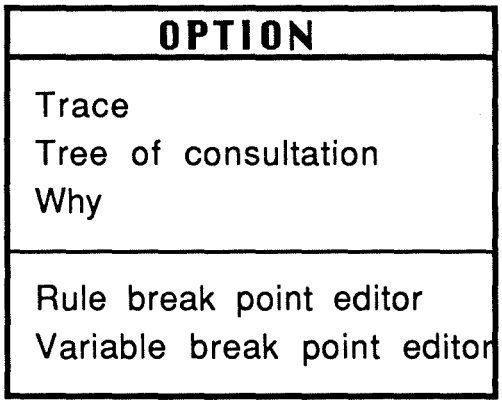


Figure 4.60 Option pop-up.

◊ Trace: Mouse-click on it to open the "last-trace" window which has been presented by figure 4.56.

◊ Tree of consultation: Mouse-click on it to open the "last-consultation-tree" window which has been presented by figure 4.58.

◊ Why: Mouse-click on it to open the "why" window. This window which possesses the presentation showed by Figure 4.61., contains explanation about the reasons for which the inference engine asks particular questions to the analyst.

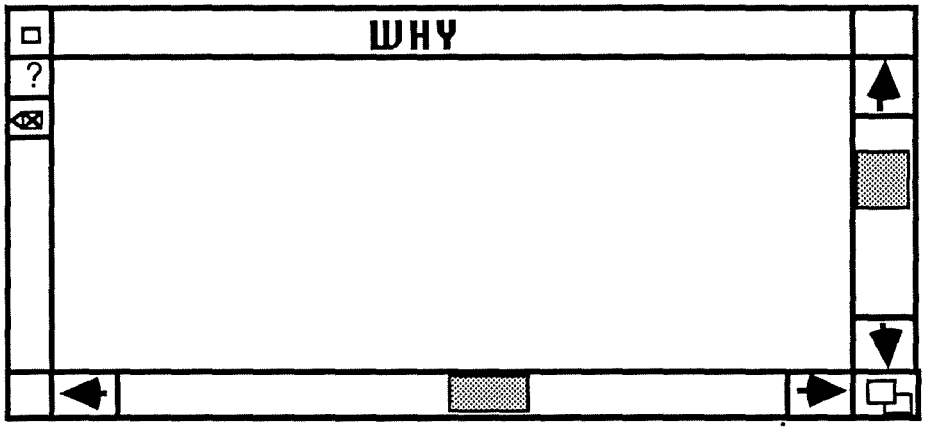


Figure 4.61: Why window.

◊ Rule breakpoints editor:

Presentation:

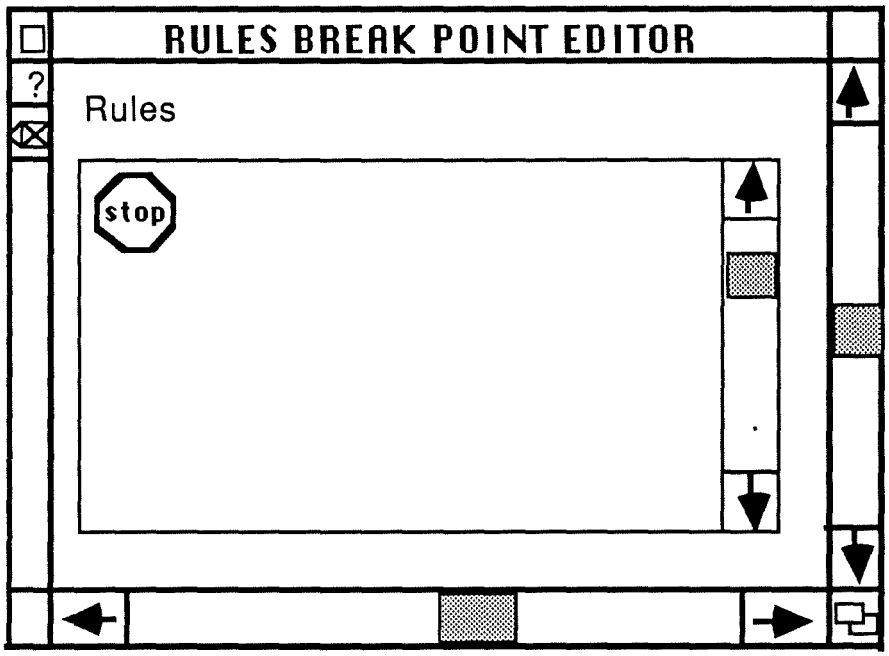


Figure 4.62: Rules breakpoint editor.

Remark: This window activated from the "options" pop-up, enables the analyst to set breakpoints on particular rules. Consequently, the consultation will stop whenever a rule, indicated as a breakpoint, is fired. This situation has an effect on the buttons. Indeed, this stopping provokes the

highlighting of the "interrupt" button and the desactivation of the "start" button.

The functioning principle is the following one:

A list of all the rules is presented to the analyst. He must click on each rule on which He wants to set a breakpoint. This action is reflected by the drawing of a "stop" icon beside the clicked rule. When the analyst has finished to set breakpoints, He just has to click on the "close box" of the window.

The removing of breakpoints consists of a second click on marked rules.

◊ Variables breakpoints editor:

Presentation:

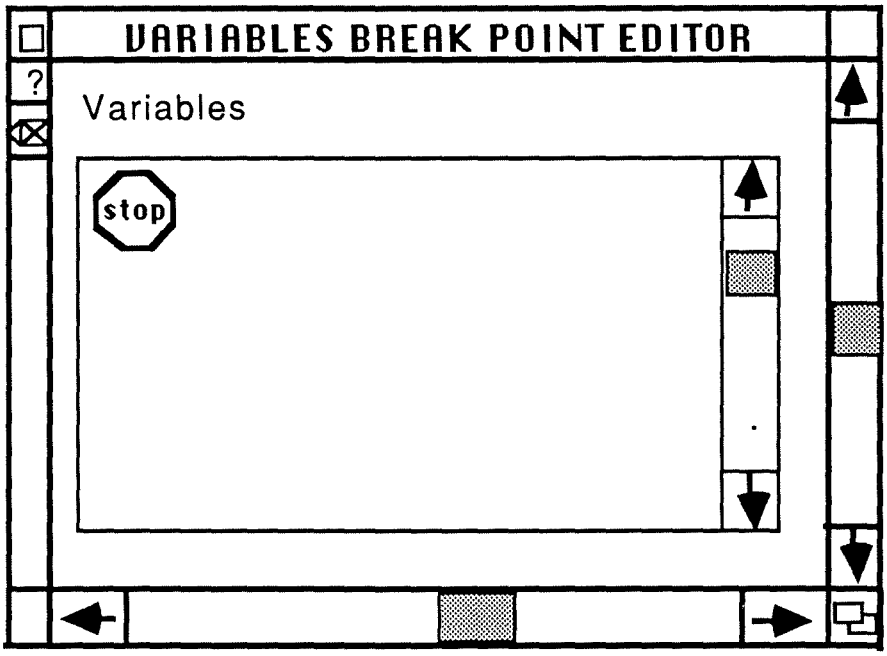


Figure 4.63: Variable breakpoints editor.

Remark: Same principles as for the previous editor, but applied to the variables.

4.13.3.5.2 "End-user-consultation" window.

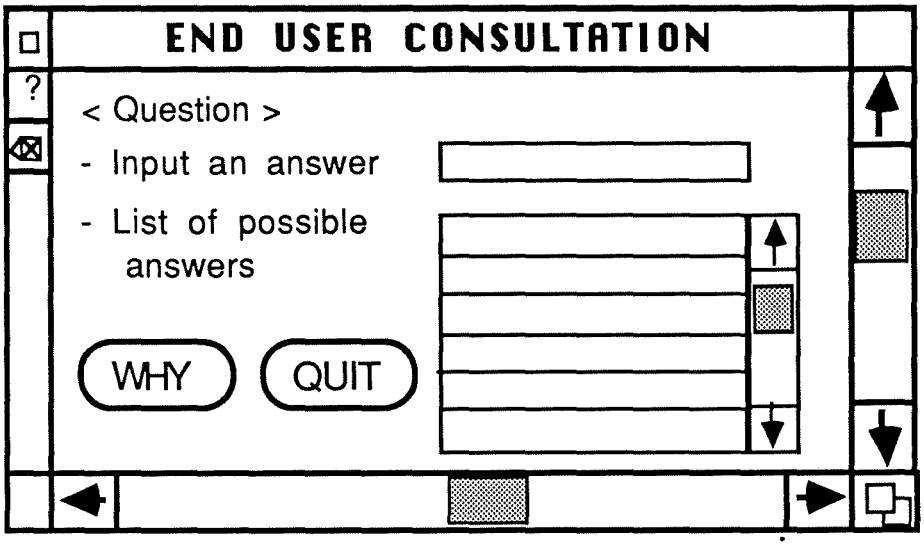


Figure.4.64: End-user consultation window

4.13.3.6 Interactive objects accessible from the "Configuration" icon.

4.13.3.6.1 Detailed presentation of the "Save-session" dialogue-box.

Name: Save session-dialogue-box.

Definition: Dialogue box enabling the analyst to save the current session under a given name.

Functionalities: The corresponding functionality is "Save-the-current-session".

Presentation:

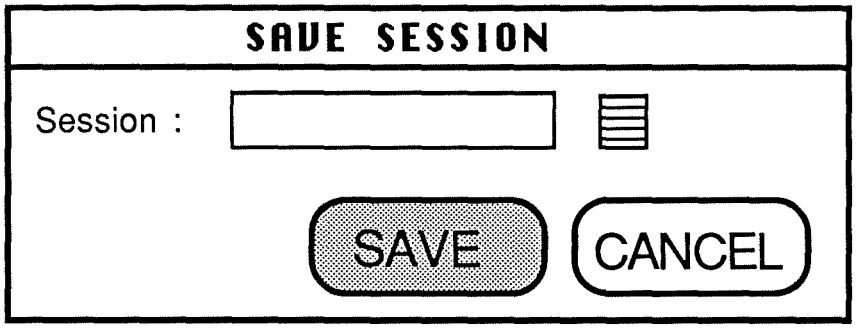


Figure 4.65: Save-session-dialogue-box.

Actions:

Window-creation: Mouse-click on the "save-session" item of the "configuration" pop-up.

Window-deletion: Mouse-click on the "cancel" button.

Window closure: Mouse-click on the "save" button or on the "return" key after the filling of the session field. Before the saving, syntactic and semantic controls are performed on the "session" field to verify its correctness. In case of problem, a dialogue box containing an error message should be displayed to the analyst asking him what to do.

Affectation of a value to the session field: Filling of the field with the keyboard of by a "list" icon.

Deletion of the session field value : Deletion of the field contents with the keyboard.

Correction of the session field value: Filling of the field with the keyboard followed by a new filling (overwriting).

4.13.3.6.2 "Restart-session" dialogue box

Presentation:

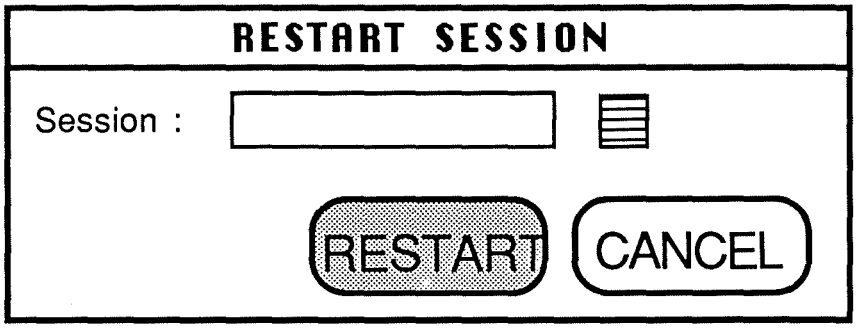


Figure 4.66: Restart-session-dialogue-box.

Remark:

Same principle as the previous dialogue box.

4.13.3.6.3 "Delete-session" dialogue box.

Presentation:

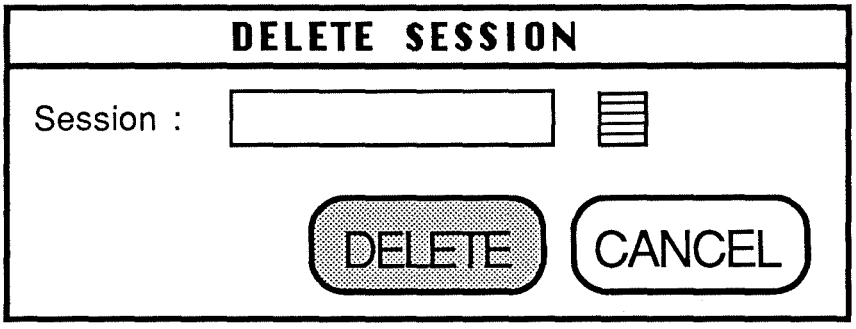


Figure 4.67: Delete-session-dialogue-box.

Remark:

Same principle as the "save-session" dialogue box.

4.13.3.6.4 "Command-language"window.

Presentation:

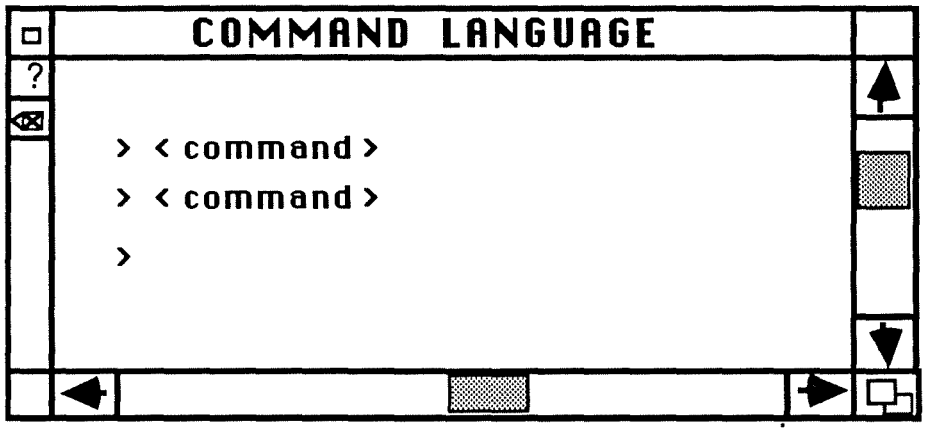


Figure 4.68: Command language window.

Remark:

The only effect of a mouse-click on the "command language" item of the configuration item is to open this window in which the analyst can type directly commands.

4.13.3.6.5 "Change parameters" dialogue box.

Presentation:

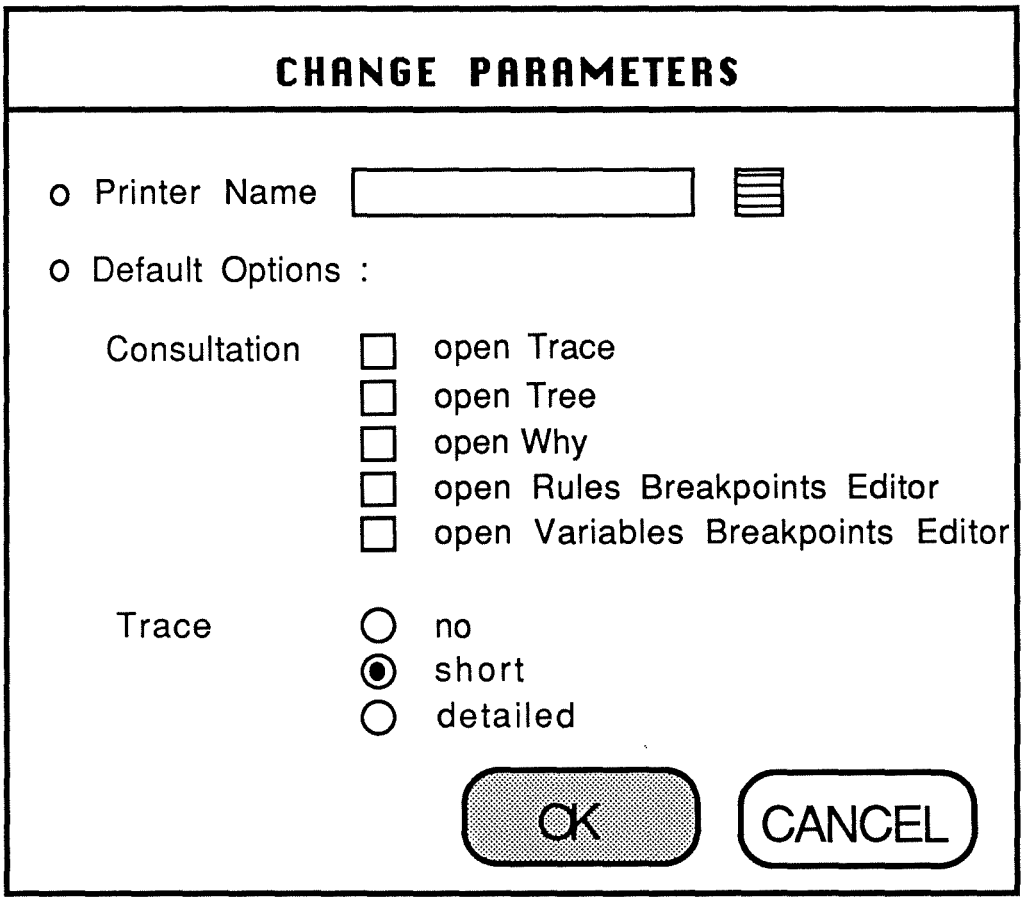


Figure 4.69: Change-parameters-dialogue-box.

Fields description:

- Printer name : enables the analyst to select or to type a printer name.

- Default options : enables the analyst to set various default options that will be taken into account when the analyst will launch a consultation and when He will access to the trace window.

4.13.3.7 Interactive objects accessible from the "Communication" icon.

4.13.3.7.1 Detailed presentation of the "Copy-expert-system" dialogue box.

Name: "Copy-expert-system" dialogue box.

Definition: Dialogue box enabling the analyst to copy an already developed expert system and all its components to another analyst or from another one.

Functionalities: Copy an expert system.

Presentation:

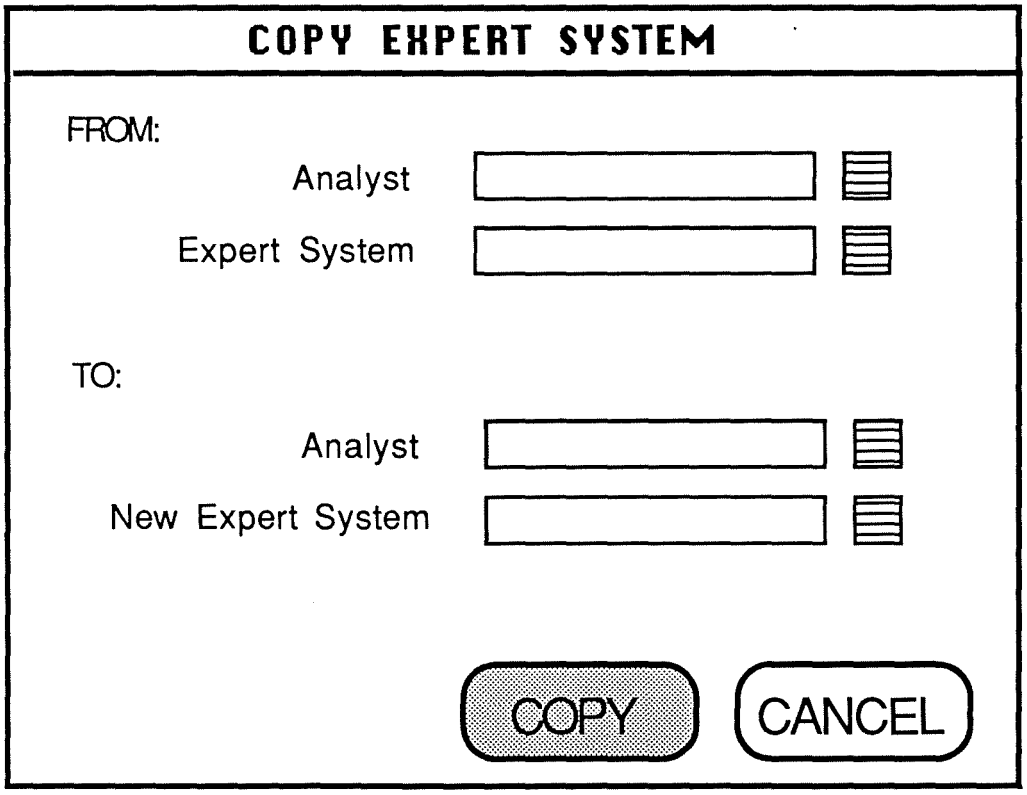


Figure 4.70: Copy-an-expert-system-dialogue-box.

Actions:

- Window creation : Mouse-click on the "copy expert system" item of the "communication" pop-up.

- Window deletion : Mouse-click on the "cancel" button.

- Window closure : Mouse-click on the "copy" button after a correct filling of the four fields.

- Affectation of a value to the fields : Filling of the field with the keyboard or by using the "list" icon.

- Deletion of the value of a field : Deletion of the field contents with the keyboard followed by a new filling (overwriting).

- Correction of the value of a field : Filling of the field with the keyboard followed by a new filling (overwriting).

Side-effects : the list icons corresponding to the expert system fields are updated automatically whenever an analyst name is put inside the linked field. Moreover it is required that the analyst introduces once its own name, but once and only once. So, when the analyst has introduced his name in one of the two analyst fields, this one is taken off out the list icon of the other fields to prevent errors. Moreover, as the analyst has always the ability to type directly in the two fields, if a problem occurs (e.g. if He has not given his name once or have done it twice), an adequate dialogue box containing an error message is displayed.

4.13.3.7.2 "Copy knowledge-module" dialogue box.

Presentation:

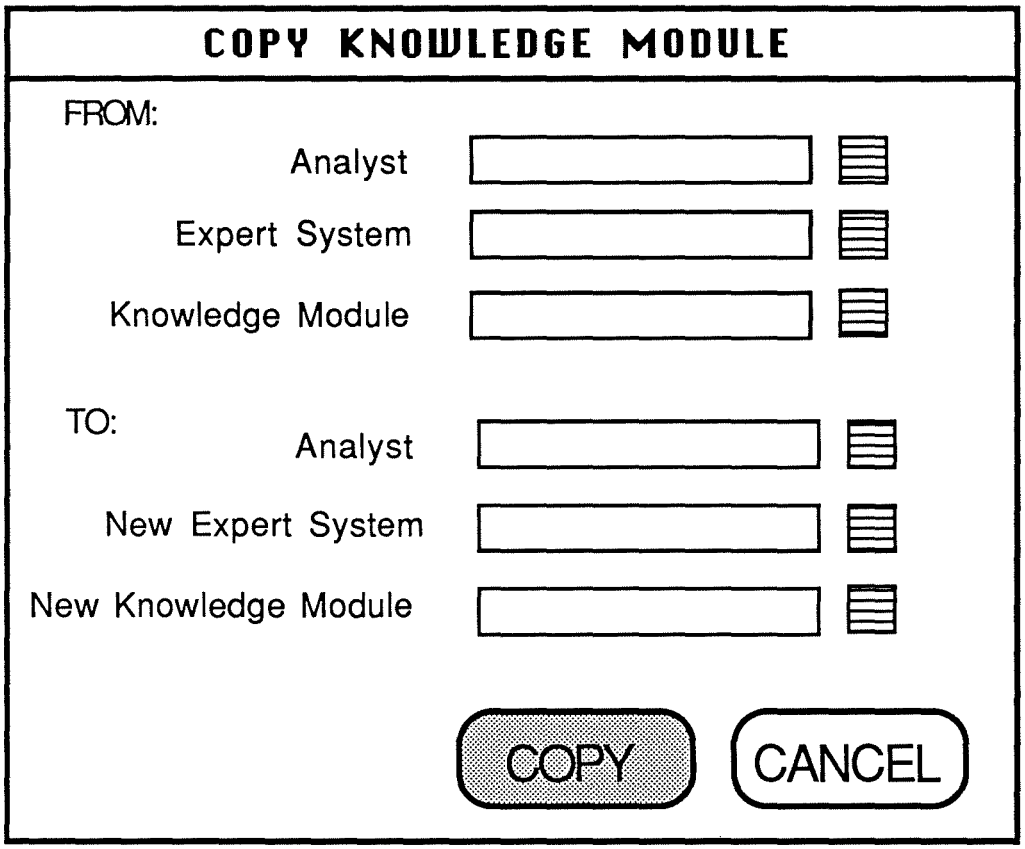


Figure 4.71: Copy-knowledge-module-dialogue-box.

Remark:

Same principles as for the previous dialogue box.

4.13.3.7.3 "Copy facts-base" dialogue box.

Presentation:

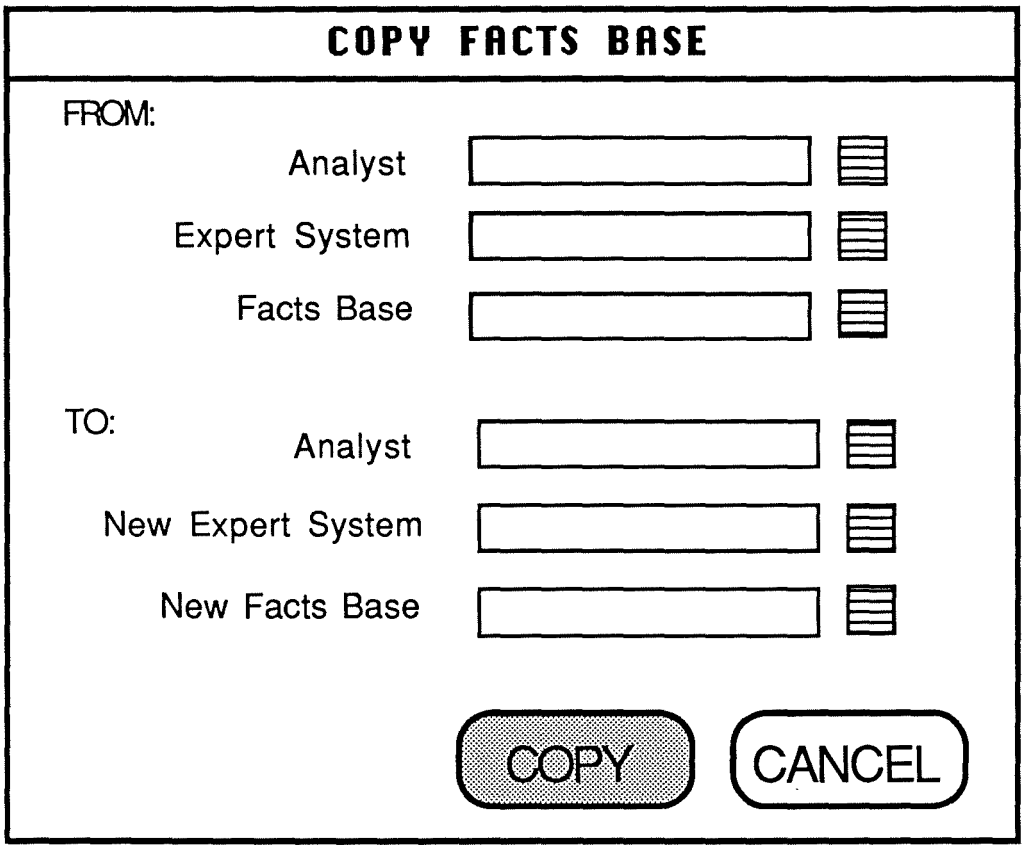


Figure 4.72 Copy-facts-base-dialogue-box.

Remark:

Same principles as for the previous dialogue box.

4.13.3.7.4 "Copy scenario" dialogue box.

Presentation:

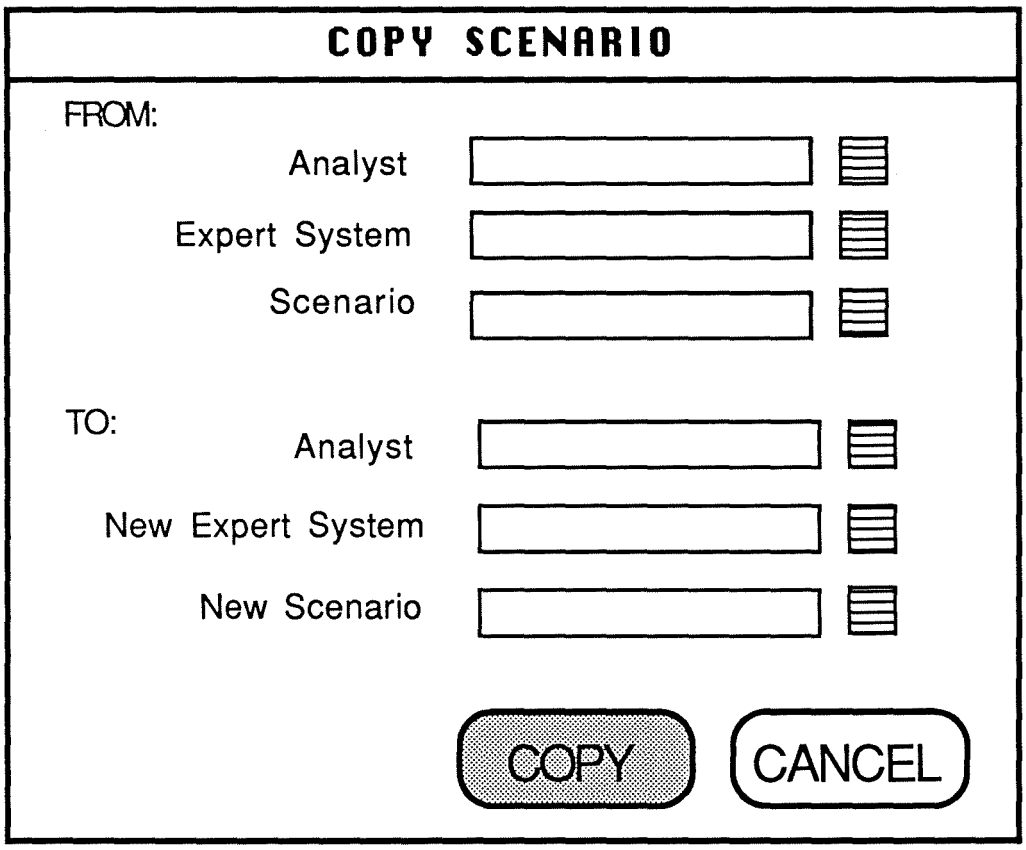


Figure 4.73 Copy-scenario-dialogue-box.

Remark:

Same principles as for the previous dialogue box.

4.13.3.8 Interactive objects linked to the on-line help function.

The last point of the specification of the interactive objects consists of the description of some typical interactive objects enabling the analyst to access to on-line help. We have said previously that the analyst may access to this kind of support at any time by clicking on the "help box" of an opened window (global help) or by clicking on an item presented in an interactive object with the right button of the mouse (particular help). This is valid through

the whole interface. Consequently, we intend to give now the specification of a global help interactive object and of a particular help interactive object associated to a particular window which is the "expert system catalogue" one.

Before it, let's underline that the global help is always displayed into a scrollable text window that the analyst can close when He has finished to read it. The particular help, on its side, is a dialogue box that is closed by clicking on an "ok" button.

4.13.3.8.1 Detailed presentation of the "Expert-systems-catalogue-global-help" window.

Name: "Expert-systems-catalogue-global-help" window.

Definition: window containing a general help text explaining the analyst the global use of the "Expert-system-catalogue" window.

Presentation:

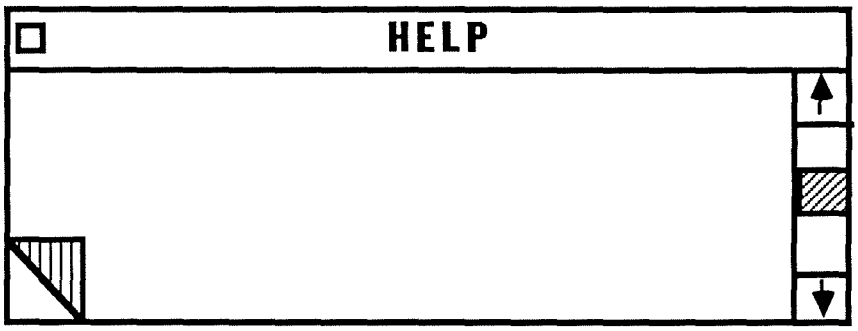


Figure 4.74: Help window.

The textual contents of the message is the following one :

" This window presents :

- *the list of all the expert system names that are available and the actions that can be performed on them.*

- *the name of the current chosen expert system. This one corresponds to the expert system which is going to be developed.*

The way to use this window is the following one :

- to select an expert system name by clicking on it with the left button of the mouse.

- then, to click on the icon symbolizing the action to perform on it.

Let's note that more information is available about each of the icons. This one may be accessed by clicking on the desired icon with the right button of the mouse.""

Actions:

Window creation : Mouse-click on the "help box" of the "Expert-system-catalogue" window.

Window closure : Mouse-click on the "close-box" of the "expert-systems-catalogue" window.

4.13.3.8.1.1 Detailed presentation of an "Expert-system-catalogue-particular-help" dialogue box.

Name: " Add-expert-system-help" dialogue-box.

Definition: Dialogue box containing an explanation of the meaning and of the way to use the icon  of the "Expert-system-catalogue" window.

Presentation:

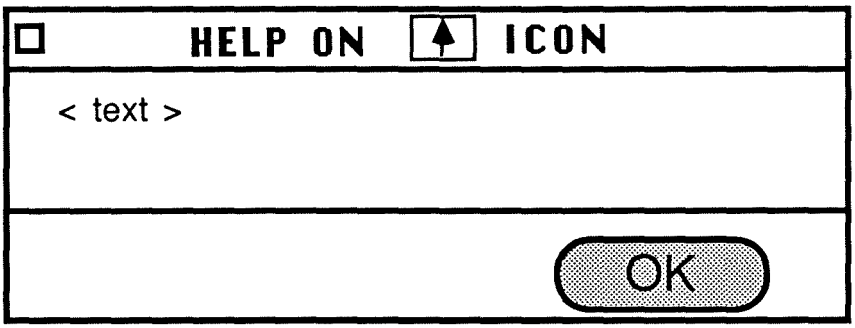


Figure 4.75: Help-dialogue-box.

The contents of the <text> is the following one :

"A click on the  *icon with the left button of the mouse displays a dialogue box in which it is possible to introduce the name and the description of an expert system. The new introduced expert system is add to the existing ones".*

Actions:

Message creation : Mouse-click on the  icon of the "Expert-systems-catalogue" window with the right button of the mouse.

Message closure : Mouse-click on the "ok" button.

4.14. CRITICISM OF OUR INTERFACE PROPOSAL

The specification of the analyst's interface we have proposed in this chapter possesses the advantage of relying on well-established and generally admitted theoretical elements among which we have kept in mind some of the most significant ones. By this way, we think it has more chances to correspond to real characteristics and abilities of a human mind involved in an interaction with a computer.

Moreover, we were particularly careful to define the considered persons and also their tasks as well as possible. Consequently, as we have started from the human intervening, we assume that the proposed interface should be really relevant from a human point of view.

The examination and use of already implemented expert system shells interfaces have attracted our attention on some interesting or, on the contrary, disturbing features of these ones. We have tried to keep these features in mind while specifying our own interface.

We have also strived to satisfy the constraints imposed by the designers of K-Expert. Among these constraints, there were notably the necessity to respect the Common User Access standard of IBM [CUA 87] as much as possible and the taking into account of a given physical environment (e.g. the interface must run on PC such as AT, XT, PS/2 and IBM compatibles equipped with a two buttons mouse).

Generally speaking, let's say that we have set the accentuation on aspects such as the analyst's freedom and control over the interface, as the visualization of the tasks objects, as the immediate and significant feedback and as the minimization of the learning and retention efforts.

This specification process has led us to experiment concretely that, as it often appears across the literature, interface design is a question of tradeoffs above all. Indeed, this remark can be illustrated by signaling that in order to preserve the interface consistency, we had to put confirmation dialogue boxes which are not really necessary under some icons (e.g. for the selection of an expert system in the corresponding catalogue). By this way, a mouse click on one of the icons of windows such as the expert systems catalogue window has always the same effect (e.g. opening a dialogue box).

Moreover, our step by step specification has emphasized the fact that the interfacing of an interactive application must be a punctilious, systematic, exacting and time-consuming operation.

The points we have just enumerated must not hide some lacks of our specification process. Indeed, the last step of this chapter concerns the specification of the interactive objects to implement in order to give a physical form to our interface proposal. The problem is that we have not reached a sufficient level of details for a future programmer. In other words, we think that a reading of this chapter is not totally sufficient to be able to implement the chaining of the specified interactive objects, for example. Always about interactive objects, let's underline that we have only used traditional ones. Nonetheless, it could be possible and perhaps more efficient to have recourse to graphic tools in the future.

We must also underline that except for the mouse configuration, we have stayed at a level relatively independent of implementation concerns. As a result, we have not precised the contents of the rules network nor the syntax and vocabulary of the recommended command language. As for the Common User Access, we have sometimes adapted it freely because it says nothing about the icons and other graphical features.

We are aware that our proposal is only a first idea of solution. As it has been shown in Section 1.4., this one should prototyped and tested on a significant panel of people in order to improve its adequation to the human reality. In this perspective, one should think especially about the quality of our analysis of the analyst's task.

Moreover, we can set the problem of the generality of the specification process we have followed in order to specify the analyst's interface for K-expert. Indeed, one may question oneselves about its validity for other expert systems and other kinds of interactive applications.

From our point of view, the retained process seems applicable to all interactive applications. However, this assertion should be submitted to a deep experimentation. Nevertheless, this is beyond the scope of our dissertation.

Finally, in order to support our specification process, a software environment should be conceived. For example, we can imagine tools able to generate automatically a prototype of the interactive messages.

CHAPTER 5 :

ARCHITECTURE PROPOSAL FOR K-EXPERT

In the previous chapters, we have concentrated ourselves on the future user of the interface we try to design. Indeed, we have presented theoretical models related to his communication process with a computer as well as guidelines to keep in mind while designing an efficient user interface. After a functional analysis of the interactive application and the determination of the profile and needs of a typical analyst, we have proposed a visualization of an analyst's interface for K-Expert.

However, this part of the work concerns only an aspect of the user interface design. Indeed, it tackles only the difficulties an analyst may encounter while interacting with a computer and it tries to bring a response to them. Nonetheless, we must not forget that the proposed interface is conceived to be implemented. So, now, abandoning the analyst's perspective, we think about the implementor's problems.

According to J. Coutaz [Cout 87], these problems belong to two categories, the architectural issues and the environmental diversity.

Architectural issues are related to the necessity for the implementor to have recourse to inputs and outputs in order to communicate information. Indeed, it appears that the need for I/O is ubiquitous and that consequently "the code which is in charge of the communication is intermixed with the code which implements the functions of the system" [Cout 87]. As a result, an interactive refinement of the user interface may be particularly intricate.

As for the environmental diversity, we can remark that the nature of the considered application imposes different requirements on the user interface. Moreover, seeing its intermediary role, the user interface may cope various users classes and also a large variety of terminals. Consequently it appears that "the construction of a user interface is a risky enterprise"[Cout 87].

To face the two evoked problems, we retain the golden rule : "Struggle for modularity"[Cout 87].

So, the aim of this chapter is to propose an architecture for the implementation of the analyst's interface which relies notably on this principle . Thanks to this architecture, the implementor should dispose of a tool to structurate his work.

In this perspective, Section 5.1. intends to summarize basic characteristics we think necessary to assign to the architecture. It also justifies our choices and recalls the elements from which we start to conceive an architecture for our particular interactive application.

Section 5.2. modelizes an architecture for an interactive application in which the dialogue control is mixed.

Then, Section 5.3 introduces a validation of our architecture by two ways. First, we underline its compatibility with the concepts highlighted in the functional analysis of K-Expert interface and with the retained UIMS (e.g. THESEUS which is presented in Appendix D.). Then, we show its efficiency by describing some typical scripts related to the processing of main functionalities of our interface.

Finally,in a concluding Section 5.4, we give a criticism of our architecture proposal.

5.1. GENERAL ARCHITECTURE FOR AN INTERACTIVE APPLICATION AND UNDERLYING CHOICES

Before considering the instantiation and the consultation of an expert system which are particular interactive applications, we think it can be helpful to define a general architecture that can be applied to every interactive application.

Generally speaking, we can distinguish three components in the architecture of such an application. This approach is inspired by [War 88c]. In this perspective, the retained components are the "Dialogue", the "Exchange" and the "Application functions". Their interrelations may be illustrated on the Figure 5.1.

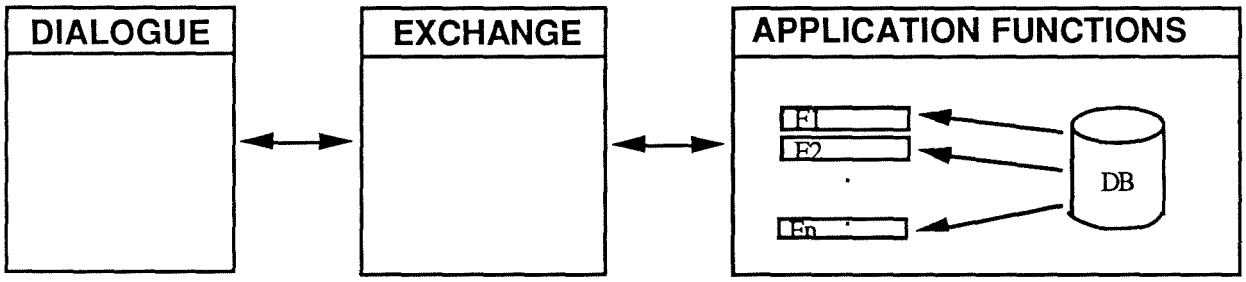


Figure 5.1 : Interactions between the architecture components.

The Dialogue component is responsible for the input (the output) of all the necessary informations from (to) the user of the application.

The Application functions component realizes the different treatments associated to the application.

As for the Exchange component, it supports the management of the communication of informations between the two previously defined components.

An important question to ask concerns the definition of the limit that can be established between the dialogue and the functions. It seems obvious that the input and the output of informations must be supported by the dialogue while the Data Base consultations are under the control of functions.

However, if it seems reasonable to affect syntactic control operations to the dialogue, it is no so direct with semantic controls because they can require accesses to the Data Base. This problem could be solved in the following way. The semantic control is performed at the level of the dialogue but this one requires the services of an application function in order to accomplish the necessary accesses to the Data Base.

In the following part of this chapter, we intend to refine this general architecture in order to adapt it to the K-Expert analyst's interface. However, there is something we can underline immediately. Indeed, this general architecture introduces the notion of independence between the application and the associated dialogue during the conception, the execution and the maintenance steps. This notion seems to us particularly critical. So, we would like to reflect it in the architecture proposal.

The importance of the independence principle relies on the following reasons.

- The conception and the implementation of the application functions and the dialogue may be attributed to different persons which are specialists in the particular associated domains (ergonomy for the dialogue, computer science for the application functions).

- The modification of one component may not affect the other one. Indeed, each component hides implementation secrets and is seen by the others as a black box. Consequently, we can imagine to modify the nature of the interactive objects used by the dialogue (for example, replace a menu-oriented interface by an object-oriented one) without having to change anything in the application functions. The converse is also envisageable : it is possible to update the contents of a function without having to reflect it in the dialogue component if the function specification remains unchanged. However, it is obvious that the independence does not imply that a fundamental modification of a component (for example, the addition of a completely new application function or feature) does not have to be reflected in the other one.

- It is possible that the dialogue takes place on a site which is different from that on which the application functions are executed. For example, one can imagine that the functions are running on a Vax configuration

while the dialogue is implemented on a Macintosh. In this case, it is the Exchange component that is responsible for the communication between the two other components.

However, we do not wish to limit ourselves to the independence principle as it has just been presented here. Indeed, we think it is necessary to put some constraints on the own architecture of the dialogue and application functions components.

It seems that in fact convenient architectures for these two components could be hierarchies of modules. By module we mean a set of functions which are characterized by a strong ability to hide information, by a strong procedural and informational cohesion and by a small coupling degree with other modules. In other words, modules regroup a set of functionalities of the same kind, they achieve a specific aim and respond to well defined specifications. Moreover, we propose a hierarchy of modules. So, retained modules should be organized in such a way that they are related in a known and ponctually way [Van 87].

Each module should be defined in such a way that when a modification has to be brought to one of its elements it is easier to throw away the module and to write it again than to modify its contents.

What is said here confirms the golden rule "struggle for modularity" [Cout 87] which has been formulated in the introduction of this chapter. As a result of what has been said about the advantages of modularity, it appears that it can be the basis for adaptation of an application or its associated dialogue. The building of a modular hierarchy for the application functions is a subject in itself. So, we do not want to insist more on it. We assume simply that a good application architecture has been designed for the whole application so that the application software possesses the following qualities : fiability, maintenability, reusability, portability, efficiency and conviviality.In the case of the dialogue and as a result of what has just been said, the general golden rule may be refined in the following manner to respond to the environmental problem.

- "Use modularity to separate functions from presentation policies" [Cout 87]. This point is acquired seeing the retained independence principle and modular hierarchies ;

- "Use modularity to define abstractions that hide the diversity" [Cout 87]. This point is also acquired if the implementation of the interactive application relies on a UIMS (as Ms-Window, for example). Thanks to such an UIMS, it is possible to build interfaces supporting the application dialogue in such a way that low level physical characteristics are hidden inside one layer. By this way, the implementation of a particular interface may be portable to various run-time environments without any change seeing that all the input/output can be expressed at a high level of abstraction.

Up to now, we have highlighted the necessity of independence and of modular hierarchies. This can be completed by a thought on the control of the dialogue. The question to be asked can be sentenced in the following manner : Who or what will be the dialogue control driver ?.

This problem of control seems to us characteristic of interactive applications. Moreover, the retained architecture for such applications should be adapted to the type of control that is chosen. Consequently, we can not let this point under silence.

At the time being, there are a lot of controversial discussions about this subject. Let's now refer to the three main streams of ideas like they are presented in [THESEUS 1] and [THESEUS 2]. The dialogue may be controlled externally, internally or in a mixed way.

The control is external when it is maintained by the UIMS itself while applications are divided into small packages, each processing one dialogue unit. It is internal if it lies under the application responsibility. In this case, the user interface may be seen as a collection of I/O services activated by the application. Finally, the control can also be considered as mixed when it is alternatively handled by the application and by the UIMS.

At present, the mixed control seems to assert itself. Indeed, internal control can be coupled with serious difficulties because "it may lead to situations where an application traps the user in a kind of local mode by forcing him to reply right away to a question without allowing a new request that would help in the choice of the correct alternative " [Cout 87]. Moreover, seeing that the dialogue is embedded in the application code itself there is no

"clean separation between mechanisms and policies" so that modifications during interactive user interface design imply changes within the application.

The external control offers a clean separation between semantics and syntax because it is the dialogue which invokes the application functions to react to particular user actions. In this case, the door is opened for the design of modular programs as the application itself may be seen as "a collection of procedures which implement the semantic actions of the dialogue" [Cout 87]. The advantage of this particular approach resides in the fact that it imposes fewer arbitrary constraints on the user than the previous one. As Coutaz summarizes it : "At the opposite of the internal control, external control quotes in accordance with a user-driven style".

By another way, the main interest of a mixed control approach lies in the flexibility it supports. Indeed, such a flexibility, which gives the implementor the ability to switch freely between internal and external controls, is particularly relevant for our case. As a matter of fact, the flexibility is very suitable for applications which dynamically require that the user inputs some more informations in order to pursue their processing as it happens during the consultation of an expert system. Another advantage linked to mixed control is the fact that it may make easier the reusability of applications. The biggest disadvantage or risk is that flexibility may go hand in hand with the introduction of dirty hacks which may provoke classic software maintenance problems. Logically, an external control with hooks for mixed control seems to be a reasonable choice. In the next of this exposure, we intend to try to take this recommendation into account.

Up to now, we have underlined a set of main principles and characteristics but it is obvious that they must be submitted to refinements in order to apply to particular cases. At the time being, studies about the research of architectural elements for an interactive application under DecWindows are in course at the "Institut d'Informatique" at Namur. The reader who wants to know more about the current state of these researches may refer to [Sac 89a], [Sac 89b], [War 88a], [War 88c].

In our case, we have used freely these studies as a base for the elaboration of our own proposal for the specification of an architecture that can be applied to a specific interactive application as the building and consultation

of an expert system. It is necessary to adapt the Namur proposals because they concern different kinds of interactive applications relative to the management while our application belongs to the domain of expert systems. The differences between these two categories of applications appear notably in the nature of the dialogue control.

For our application, we retain an external control with hooks for mixed control. We mean that the control is always under the responsibility of the UIMS except when the latter invokes an application which takes the control in order to receive additional informations from the user. Our choice relies on the fact that we have decided to see our application as a tool box of functions triggered off at the analyst's request. So, in this perspective, the envisaged control should be external. Nonetheless, hooks to internal control must also be provided in the special situation when the analyst decides to run a consultation. The applications envisaged at Namur, however, present another kind of mixed control. We can consider it as an internal control with hooks for mixed control. Indeed, the considered interactive applications possess a dynamic defining the chaining of the application functions. The control is transmitted to the UIMS only if one of them needs some external information to fulfil its specification. In this case, the UIMS is triggered off and keeps the control until all the needed informations have been gathered.

Moreover, these proposals are related to the UIMS DecWindows when the UIMS we must consider is THESEUS. Consequently, in the next pages, we think about the adaptation of Namur architectural elements.

So, now in a first time we are going to define a general architecture that may be applied to an interactive application led by a mixed dialogue control. Then, we intend to precise the nature of THESEUS UIMS to provide an empirical validation of our proposition.

5.2. MODELIZATION OF THE ARCHITECTURE OF AN INTERACTIVE APPLICATION

Let's now present an architecture which relies on modularity in order to respect the general golden rules already presented. As constitutive

interface modules, we retain the interactive objects manager and the conversation manager.

As constitutive application modules, we propose the application functions module and the Data Base Management module. The application functions module may itself be an architecture of several modules but this is beyond the scope of this work. So, we consider the whole application functions module as a black box.

Finally, to preserve the independency between the interface and the application, our architecture introduces an interactive application manager. All these modules will be linked together by two kinds of relations, the so-called "UTILIZE" and "CALL" relations.

So before giving a specification of each component module, let's define these two relations. A UTILIZE relation between two modules A and B means that the correct working of the module A depends on the availability of a correct version of the module B. A CALL relation between two modules A and B means that the execution of a treatment belonging to the module A triggers off the execution of a treatment belonging to the module B. The difference from the utilize relation is that, in a call relation, the module A may work correctly even without disposing of a correct version of the module B [Van 87].

The architecture we present here is a general one. It can be applied to each interactive application but we often emphasize on our particular case : the instantiation of K-Expert shell and its consultation. To specify this architecture, we give the definition and objective of each module, the required inputs, the outputs it provides and also the relations UTILIZE and CALL linking it to each of the other specified modules.

First of all, let's present a visualization of our interactive application architecture proposal by the following Figure 5.2.

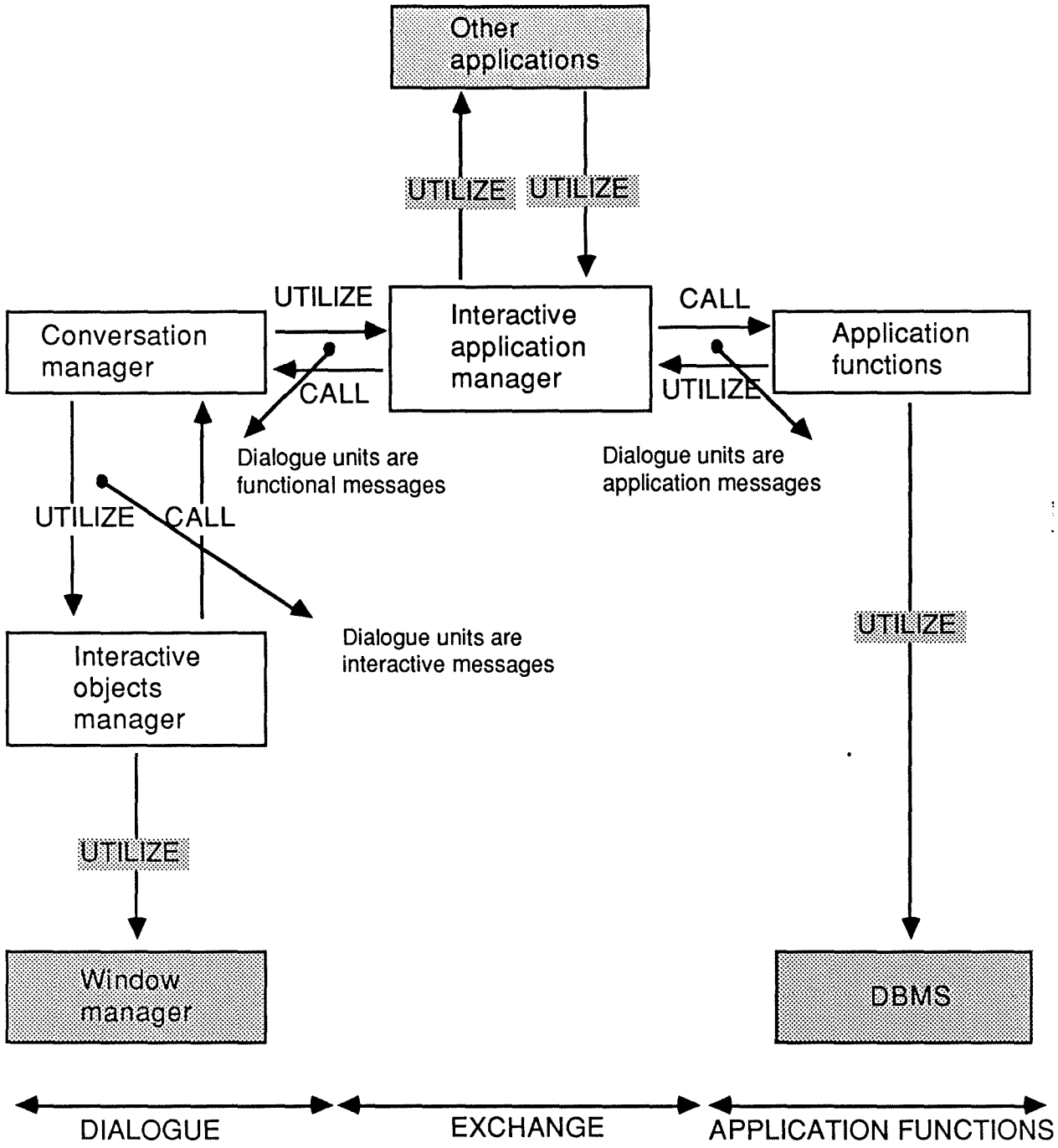


Figure 5.2 : Architecture proposal.

The modules presented in dotted boxes are not detailed in this work. They are just mentionned in order to localize them with regard to the other ones.

5.2.1. The interactive objects manager module

◊ Definition :

This module is a high-level input-output mechanism.

◊ Objective of the module :

This module makes possible the hiding of the device specific input-output functions to the other modules. Moreover, it enables them to perceive input and output from a higher level of abstraction.

Thanks to this modules, it is possible to express input-output by recourse to the abstractions provided by interactive objects. We have already defined the concepts of interactive objects as instantiations or compositions of instantiations of generic objects. These generic objects are graphical ready-to-use objects possessing properties which can be inherited by their instantiations. By this mechanism, the other modules can request the display of a whole dialogue unit meaningful for an user in once time. This unit is described in an interactive message. So, when a low level input arrives from the system to this module, the latter expresses it for the other modules as an action on an interactive object. On the other side, when an interactive object must be displayed, the considered module translates this high level output into a low level output understandable by the underlying window manager or by the operating system if there is no intermediary layer. Concretely, this module corresponds to the currently available UIMS. As for the input/output abstraction level, it depends on the power of the considered UIMS.

In the following of this Section, we are going to describe the function of the interactive objects manager which consists of the translation of high-level outputs into low level outputs. Then, we explain the other function which consists of the translation of low level inputs into high level inputs.

◊ Output translation function :

Input data :

An input interactive message with the following contents :

- A description of the interactive objects composing the dialogue unit to display ;
- For each object, a description of all the possible actions that can be performed by an user on it ;
- For each action associated to an object, the name of a function that must be called if the user performs this action. However, it is not always mandatory to designate an associated function ;
- If the interactive object is an input field, a description of the syntactic constraint on the pattern contents that is legal for this field. By "pattern contents", we mean the specification of the authorized characters such as numeric, alphanumeric ones. This syntactic constraint is optional. A text can also be associated in order to be displayed in case of an user mistake.

Preconditions :

- The described interactive objects are instantiations of generic objects known by the module ;
- The given actions are instantiations of generic actions that can be accomplished on the specified interactive objects ;
- The associated function names correspond to functions available in the conversation manager module. This module is explained more in details in this section ;
- The associated pattern is one of the patterns known by the module.

Output data :

- Displaying on the screen of a dialogue unit corresponding to the given interactive message.

Postcondition :

All the described actions on the interactives objects are available for the user.

◊ Input translation function :

Input data :

- An occurrence of an event signaling that an user has performed an action on a displayed object.

Precondition :

There is no precondition on this input because this module must take into account all the inputs coming from the user.

Output data :

- Updating of the screen ;
- Display of an error message ;
- Production of an interactive message with the following contents :
  - . the name of a function of the conversation manager module which has to be triggered off ;
  - . parameters indicating the actions performed, the object concerned, and other ones that are needed by the triggered off function. For example, if the action is the filling of an input field, a parameter must contain the input value.

Postconditions :

- If the performed action is an action on a property of a generic object which is related to "local dialogue" (e.g. action affecting only the

presentation of the interface such as resize a window, move a window, move a scroll box...etc) then this module treats the event by itself and updates the screen ;

- If the performed action corresponds to the filling of an input field, this module verifies if the input contents is consistent with the related syntactic constraint, if there is one. If the field is not syntactically correct, an error message is displayed. If the error message has been specified, this one is displayed otherwise, a default one is shown off ;

- If the performed action is related to the semantic of the application (e.g. if the event is relevant for the application or for the management of the dialogue), needs particular treatment and is syntactically correct (in the case of an input field) then, this module produces an output interactive message. In this message :

  . the given function name corresponds to one function of the conversation manager which is defined in the next section;

  . the other parameters correspond to objects and actions known by the conversation manager ;

  . in the case of an input field, the value is syntactically correct.

◊ Relations with other modules :

The interactive object manager CALLS the conversation manager module. Indeed, the interactive object manager calls the conversation manager whenever it has performed all the treatments it could realize by itself. So, the bad functioning of the conversation manager can not affect the work of the interactive object manager.

5.2.2. The conversation manager module

◊ Definition :

This module controls the dialogue between the user and the interactive application.

◊ Objective of the module :

This module is responsible for the dynamic of the dialogue of the application. It contains all the necessary dialogue functions for the interactive application. It hides the other modules the way according to which the dialogue is managed. The choice concerning the interaction style with the user (such as menu interaction and direct manipulation) is a secret of this module. It decides the behaviour to adopt in order to ensure the continuation of the dialogue with the user. It knows which interactive message (e.g. user dialogue unit) must be displayed or which service must be required from the application. In this case, this module is able to build the contents of the message that must be given to a function. So, we can say that this module manages the so-called "macro conversation". By this, we mean the chaining of the interactive messages between them.

As an interactive message may be represented by one or more interactives objects, this module knows which are these objects and the so-called "micro conversation" associated to them. By "micro conversation", we mean the chaining of the interactive objects to display in order to embody an interactive message. This module is also able to determine if an user's input is syntactically (high-level control) and semantically correct. In case of problem, it knows also which interactive error message has to be displayed.

As we have made the hypothesis that the control of interactive applications is a mixed one, the control can be led by the actions accomplished by end-users on the interface or by the application itself. So, the considered module can be called by two other modules. Indeed, when the control is external (e.g. depending on user actions on interactives objects) this module is called by the interactive objects manager. On the other hand, when the control is internal (e.g. handled

by a function of the application) this module is called by the corresponding function via the interactive application manager (this module is defined later).

Let's now define the inputs and outputs of this module for the two kinds of control.

◊ External control :

Input data :

An output interactive message with the following contents :

- The name of the function of this module which must be executed ;
- The input parameters necessary to be able to trigger off the execution of the function. Among them, there are the action performed by the user, the objects on which it has been realized, the value typed into an input field, the state of an object...etc.

Precondition :

The given function name is known by the considered module. The other parameters such as the given action and the given object are also known by the module. Moreover, if one parameter gives the value of an input field, the format of the given information is correct.

Output data :

An input interactive message with the following contents :

- A description of the several interactive objects which must be added, updated or deleted on the screen ;
- For each action associated to an interactive object, a name of function to call if the user performs this action. This function reference is optional ;

- If the interactive object is an input field, a description of a syntactic constraint on the format of the contents that can be typed by the user. By "format", we mean the specification that the contents must be numeric, alphanumeric...etc.

Postconditions :

- If the module is called because the user has filled an input field, this module will verify if the typed information is syntactically and semantically correct. Indeed, the given information is already conform to the desired format but other syntactic controls have to be performed. They are :

  . value control (the value of a field ranges between two constants);

  . type control (the value respects a special syntax defined for this type of value. For example, the syntax of an antecedent and of a consequent of a rule are well-defined and must be respected) ;

  . existence control (the input field must have a value) ;

  . other controls can be performed : the semantic ones. For example, it is possible to verify if the given value is really an identifier.When the typed input is not syntactically or semantically correct, the module produces an input interactive message. This message describes the interactive object and the text to display in order to signal the error to the user. The advantage of putting a control at this level is that it is helpful to detect errors at the field level and it provides the user the ability to realize immediately corrections.

- If, in order to continue the dialogue, the module needs either to add or to modify or to delete interactive objects on the screen. So, it produces an output interactive message. But in order to produce this interactive message and to make possible the continuation of the dialogue, this module may need the result of the execution of services provided by the application. For example, it is useful to

perform semantic control or intricate syntactic control, to display information recorded in the Data Base, to execute some application function asked by the user..etc. In this case, the module will produce an input functional message with the following contents :

. the name of a service of the application which has to be provided ;

. all the input parameters necessary for the execution of the service ;

. all the output parameters which have to be produced by the execution of the application service ;

This message is intended for the interactive application manager module which is described later. The given name is known by this module and the parameters correspond to necessary input and output of this service.

The external control evoked here can be modelized by the following Figure 5.3 :

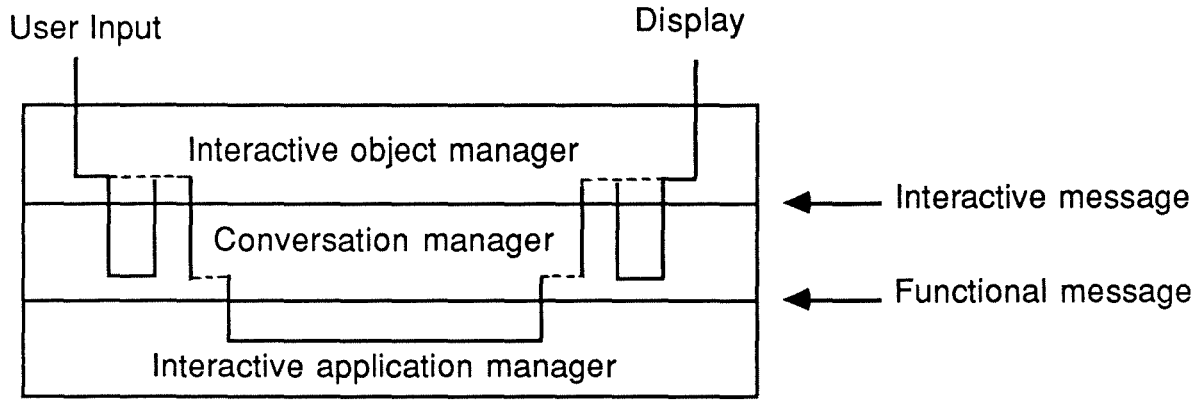


Figure 5.3 : External control in the conversation manager.

◊ Internal control :

Input data :

An output functional message with the following contents :

- The name of the function of this module with has to be executed ;
- All the necessary input parameters for this function ;
- All the output parameters which have to be produced by the function.

Precondition :

The given name of the function is known by the module. The given parameters are correct relating to the precondition corresponding to the function.

Output data :

Filling of all the output parameters of the output functional message and production of a signal to the application indicating the functional message is correctly filled.

Postconditions :

All the filled parameters are syntactically and semantically correct. In order to fill these parameters, the conversation manager needs to drive a dialogue with the user. It has to display dialogue units to the user and to enable him to perform actions on them. This dialogue is a temporary one and exists until the output functional message is correctly filled. However, during this dialogue, the interactive application is led by the user action performed on the displayed objects. So, we can say that a temporary external control is started.

During this temporary external control, the conversation manager will have the same behaviour as that already described in the paragraph about the external control. The internal control can be modelized by the following Figure 5.4 :

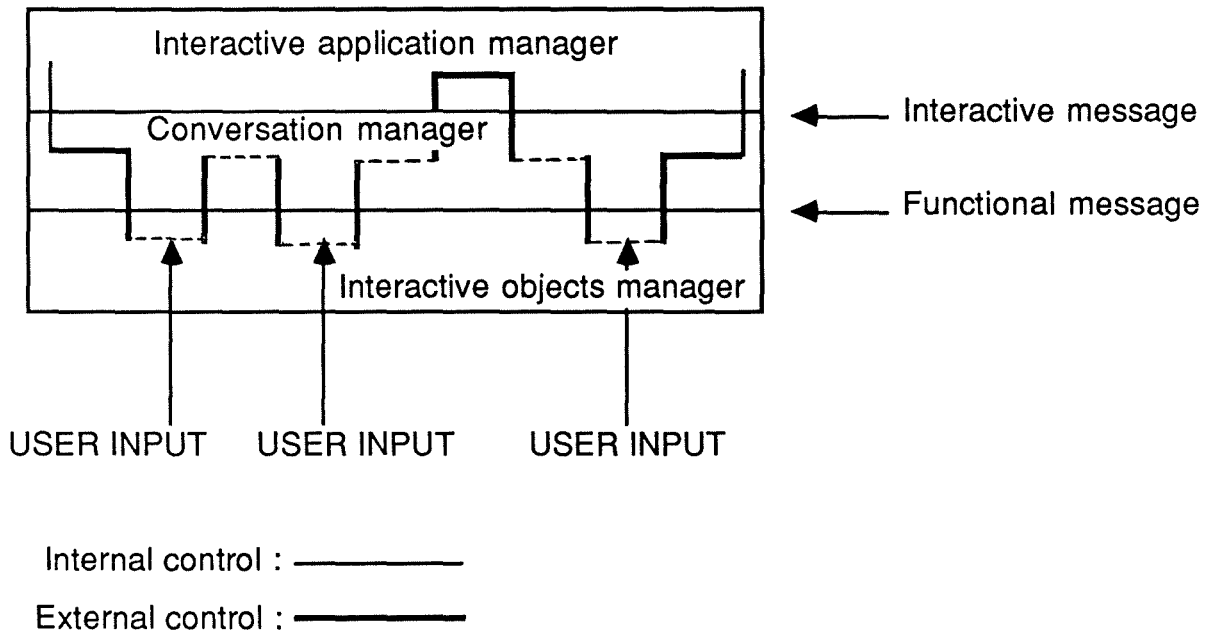


Figure 5.4 : Internal control in the conversation manager.

◊ Remark :

Seeing that the interactive application manager module has not yet been described, we signal that this module is responsible for the production of output functional messages.

◊ Relations with other modules :

The conversation manager UTILIZES the interactive object manager. Indeed, if the interactive objects manager does not work properly when either it displays interactive objects or triggers off other functions of the conversation module for a particular event, the conversation module will be unable to manage the dialogue of the interactive application.

In the same way, the conversation manager UTILIZES the interactive application manager module. Indeed, this module enables the conversation manager to receive the result of the execution of

application functions. If the desired results are not correct, the dialogue of the interactive application can not be managed.

5.2.3. The interactive application manager module

◊ Definition :

This module is the manager of the interactive application. It decides the control which must be the leader at the beginning of the execution of the interactive application and implements the transparence between the requests of the application modules and the services offered by the interface and conversely.

It holds the mapping between the abstract world of the application functions and the concrete world of the interface. Moreover, this module is also an entry point for all other applications wishing to use some of the functionalities proposed by the interactive application. If necessary, it can also be on exit point for the request of functionalities available in other applications.

◊ Objective of the module :

This module decides of the control of the interactive application. Indeed, in the interactive application, the control is always a mixed one but it can be of two different kinds :

- An external control with hooks to internal control. In this case, at the beginning of the interactive application the module gives the control to the conversation manager module ;

- An internal control with hooks to external control. In this case, at the beginning of the interactive application the module gives the control to the application functions module (which is described later) ;

Another aim of this module is to implement the transparence between the interface and the application. Indeed, during an external control, when

the conversation manager, in order to pursue the dialogue, needs the result of a service provided by the application, it gives to this module an input functional message. The contents of this one precises the name of the desired application service. The given name is not necessarily the real name of the desired application function which has to be triggered off and moreover, the asked service may require the execution of more than one function. In this case, the interactive application module makes the translation and triggers off all the necessary functions in order to produce the asked informations.

By another way, during an internal control, when an application function needs some user input in order to reach the aim corresponding to its specification, this function asks the interactive application manager to start a dialogue with the user. This module translates this request into an output functional message understandable by the interface and more particularly by the conversation manager module which has to perform the desired dialogue.

So, it appears that the interactive application manager gives to one module the list and the specifications of the available functionalities realizable thanks to the execution of functions of the other module. And it enables this module to consider the other module as a black box.

Moreover, this module is an entry point for all the other applications. Indeed, an other application can need to execute one of the functionalities of the application functions module. In this case, the calling application will have the same behaviour as the conversation manager, it produces an input functional message and calls the conversation manager module.

On the other hand, another application can need to use functionalities of the interface. In this case, its behaviour will be the same as that followed by the application to start a dialogue with a user. We can also imagine that the interactive application module enables the interface modules and the application modules to request services of some external functions.

As the presented module takes in charge the mapping between the interface and the application, we present first the translation function

"interface-application" which occurs during an external control and then, the translation function "application-interface" which occurs during an internal control.

◊ External control ("interface-application" translation function) :

Input data :

An input interactive message with the following contents :

- The name of a service of the application which has to be provided ;
- All the input parameters necessary for the execution of this service ;
- All the output parameters which have to be produced by the execution of the application service.

Precondition :

- The given name is known by the interactive application manager ;
- The given parameters correspond to the necessary ones for the execution of the service.

Output data :

Filling of all the necessary output parameters of the input functional message and production of a signal to the calling module (the conversation manager) indicating that the input functional message is correctly filled.

Postconditions :

- All the parameters are correctly filled in order to satisfy the conversation manager request, the interactive application manager calls one or more functions of the application module. It provides them input application message (the content of this messages and the description of this module are described later). The external

control during the considered translation function may be illustrated by Figure 5.5. .

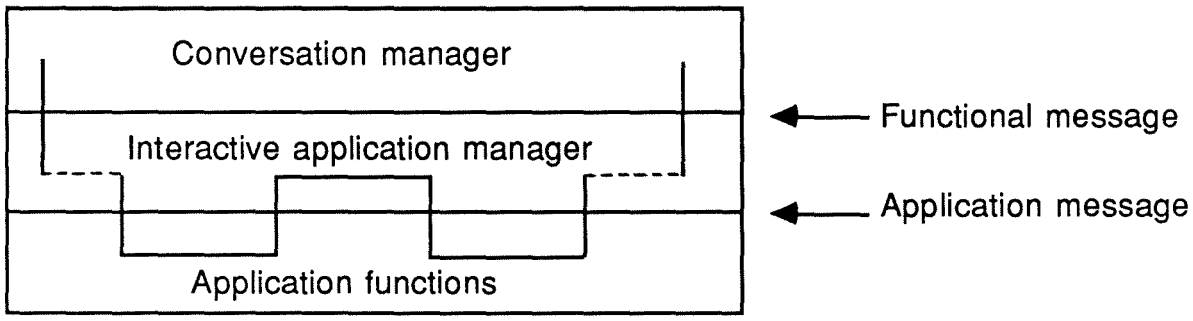


Figure 5.5 : External control in interface-application translation.

◊ Internal control ("application-interface" translation function) :

Input data :

An output application message with the following contents :

- The name of a service of the interface which must be executed ;
- All the input parameters necessary for the execution of the service ;
- All the output parameters which have to be produced by the execution of the interface service.

Precondition :

The given name is known by the interactive application manager module. The given parameters correspond to the necessary ones for the execution of the service.

Output data :

Filling of all the necessary output parameters of the output application message and production of a signal to the calling module (the application function module) indicating that the output application message is correctly filled.

Postconditions :

All the output parameters are correctly filled. In order to satisfy the application function request, the interactive application manager calls the conversation manager module and provides it with an output functional message (the contents of this message has already been described in the input of the conversation manager module).The internal control during the considered translation function may be illustrated by Figure 5.6. .

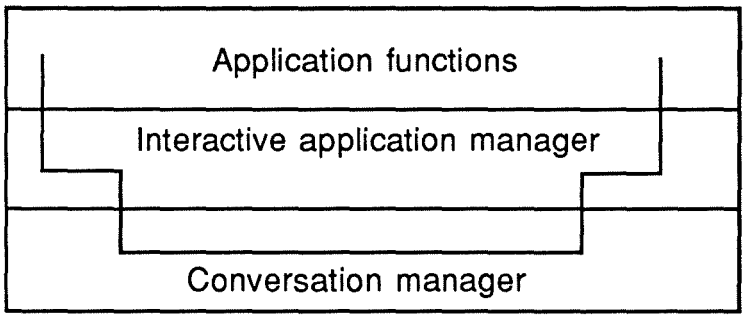


Figure 5.6 : Internal control in interface-application translation.

◊ Relations with other modules :

The interactive application manager CALLS the application functions module. Indeed, if the application functions module does not work correctly, as the considered module does not really perform treatment on the manipulated informations but only fills a functional message starting for an application message, it can always accomplish its job.

The interactive application manager CALLS the conversation manager module. This can be explained in the same way as for the previous

evoked relation because this time, the module fills application messages starting from a functional message. So, the returned values of the conversation manager have no meaning for it.

5.2.4. The "application functions" module

◊ Definition :

This module contains all the functions of the application. Generally, these functions are related to the structure of the Data Base but some of them can also be calculation ones.

◊ Objective of the module :

This module contains a set of functions which are able to implement all the functionalities necessary by the interactive application. This module can be organized as an architecture of modules in which each module hides a particular secret.

Input data :

An input application message with the following contents :

- The name of the application function which must be executed ;
- All the input parameters necessary for this execution ;
- All the output parameters which must be produced by this function.

Preconditions :

- The given name is knows by the module ;
- The given parameters correspond to the necessary ones and are syntactically and semantically correct. They must verify all the preconditions of the desired function.

Output data :

Filling of all the necessary output parameters of the input application message and signaling to the calling module that the functional message is correctly filled.

Postcondition :

The output parameters are correctly filled. To fill it, this module may use the Data Base management module. But the functions are built in such a way that they preserve the consistency of the data structures.

Some of the functions need interactions with the user in order to reach their specification. In such a case, the module utilizes the interactive application manager to obtain the necessary informations. In this case, the application function produces an output application message which has already been defined in the input part of the interactive application manager. The external control in the considered module may be illustrated by Figure 5.7.

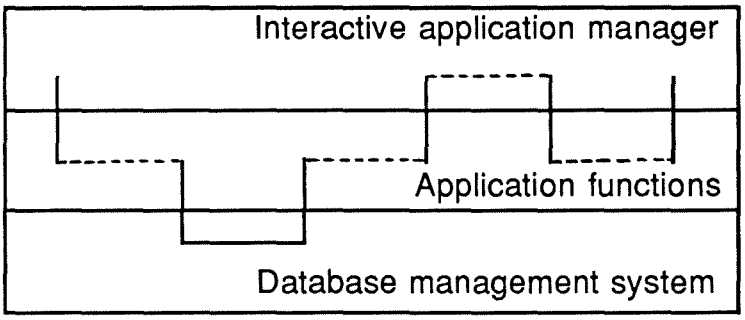


Figure 5.7 : External control in the application functions manager.

◊ Relations with other modules :

The application functions module UTILIZES the Data Base management system because the application functions are only able to carry on in conformity with their specification if the modifications performed on the Data Base are well done.

The application functions module UTILIZES the interactive application manager. Indeed, this module enables the application manager to lead a dialogue with an user and to receive user informations. If the dialogue can not be performed correctly, the application function is not able to reach its specification.

5.2.5. Architecture illustration

To conclude, we can give an example of an execution of an interactive application through our proposed architecture.

Hypothesis : The considered interactive application is led by external control with hooks to internal control. Its process may be visualized on the following Figure 5.8. .

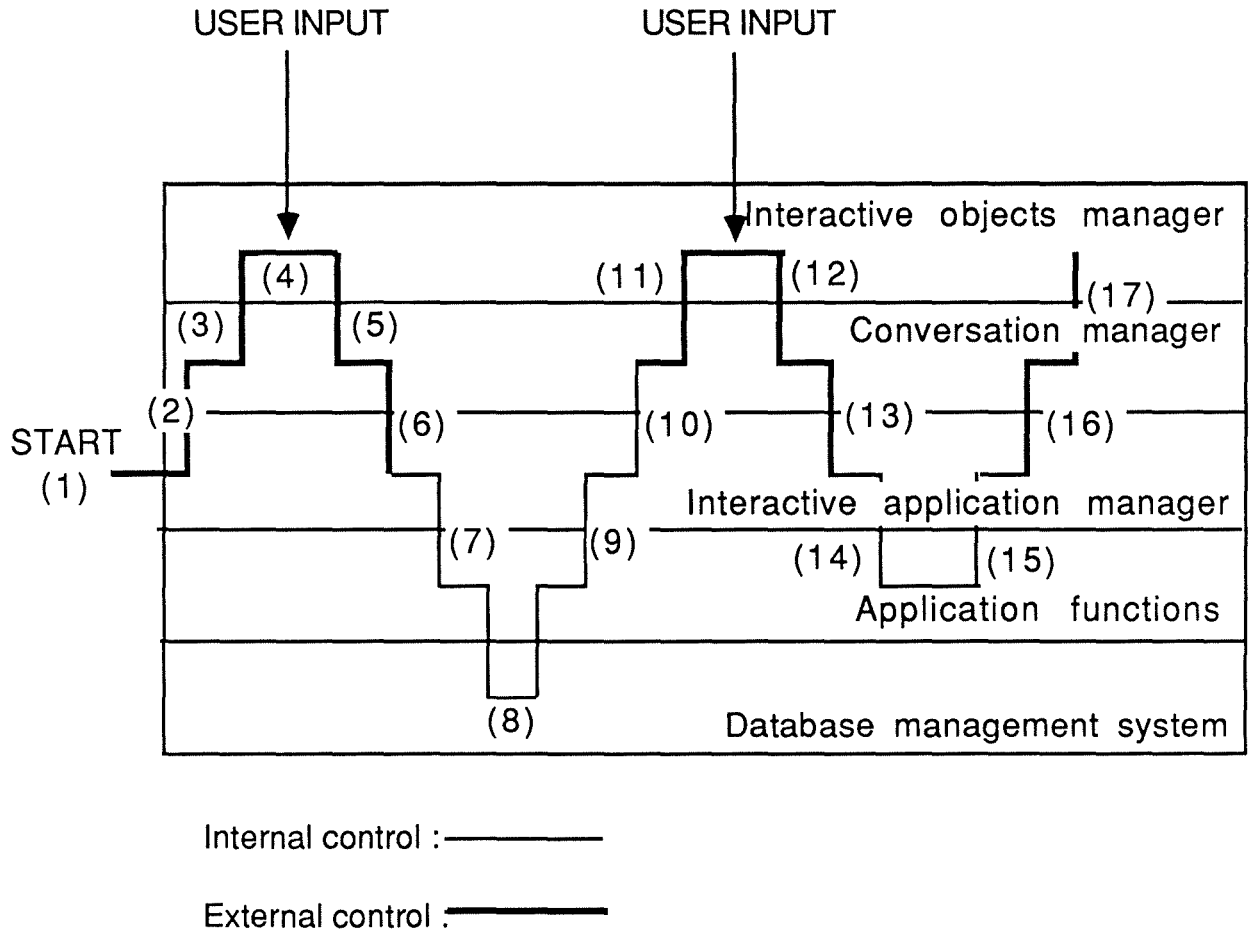


Figure 5.8 : Architecture illustration.

(1) The control is started in the interactive application manager. This module decides if the application control is an internal one or an external one. Following our hypothesis, this control is external.
So, (2) the module calls the conversation manager and provides it with an output functional message.

As the conversation manager needs to display some interactive objects on the screen, it calls the interactive objects manager and provides it an input interactive message (3).

The interactive objects manager displays the necessary interactive objects. The user performs an action on one of the displayed objects (4).
The interactive objects manager calls the conversation manager and provides it with an output interactive message (5) containing the name of a dialogue function to execute.

(6) This dialogue function of the conversation manager may require some Data Base informations in order to display new interactive objects on the screen (such as a list of possible values). So, it produces the corresponding input functional message and calls the interactive application manager. From this point, a temporary internal control is started.

Indeed, the interactive application manager module decides to give the control to an application function of the application (7). It provides an input application message and calls the application functions manager.

(8) The application function has to fetch information from the Data Base. If the application needs also some more information from the user, it must ask to start a dialogue with the user. So, it provides an output application message and calls the interactive application manager (9).

(10) The interactive application manager produces the corresponding output functional message and calls the conversation manager. At this point, once again, the control changes and becomes a temporary external control.

(11) In order to fill the output parameters of this message, the conversation manager asks by interactive messages to display some interactive objects. It triggers off the interactive objects manager.

(12) The user performs an action. The conversation manager is once again triggered off and it receives an output interactive message.

(13) The conversation manager is now able to fill all the output parameters of the functional message it has previously received. Then, it calls back the interactive application manager.

The temporary external control is ended at this point. (14) In his turn, the interactive application manager is able to fill the output parameters of the output application message it has received previously also and it gives back the control to the application function manager.

(15) This application function is now provided with all the informations it needs to performs its task. Then, it can also fill the output parameters of the input application message it has received and it gives the control back to the interactive application manager.

(16) This time, the interactive application manager is able to conclude the input application message and to give it back to the conversation manager. The temporary internal control is concluded at this point.

(17) The conversation manager is now able to display some new interactive objects. It produces an input interactive message that is given to the interactive objects manager.

5.2.6. Final remark about the presented architecture

In our architecture, the UNDO function has never been mentioned. However, this one should be taken into account. The advantages of providing this function to a user have already been defined in Chapter 1.

The common definition of the UNDO is to provide the ability to recover from unwanted or incorrect actions. An UNDO function, which reverses the effect of the last action performed can be used to provide such an ability. However, the problem is that the need for reversible actions applies at many levels and should be taken into account by differents modules of our proposed architecture.

Indeed, the last action of a user can be the dragging of an object or the resizing of a window. The management of this action is only performed by the interactive objects manager and is transparent to the other modules. So, if we enable the user to make an UNDO on this kind of actions, the function must be supported by the interactive objects manager.

Another level of UNDO concerns the chaining of the interactive messages. If a user clicks on an item of a pop-up menu and if the result is to open a new window, then the UNDO of such an action would be to delete the window of the screen and to bring back the user to the previous pop-up menu. This UNDO should be taken in charge by the conversation manager because it is the only module to be busy with the macro conversation.

This module must also take care of the UNDO function for the micro conversation. Indeed, if the filling of a field triggers off in the same window the

updating of a list, the UNDO of this action has not only to give back the previous value of the field but also all the previous values of the updated list.

There is still a higher level of UNDO. If the last action performed by the user has triggered off modifications of the Data Base, an UNDO of this action should be able to restore the previous state of the Data Base. This time, the conversation manager should be able to trigger off functions of the application restoring the Data Base contents.

To conclude, we can say that the UNDO must be taken in charge by two modules. The interactive objects manager and the conversation manager. Moreover, in the application functions module, for each functionality which can be called by the conversation manager in order to perform modification on the Data Base, other functionalities have to be provided in order to enable the conversation manager to restore the state like it was before the last action.

5.3 VALIDATION OF THE PROPOSED ARCHITECTURE

In absence of the implementation of the K-Expert interface proposed in Chapter 4, it is not possible to validate concretely the retained architecture.

So, what can be envisaged is a more empirical validation. Indeed, we are now going to show that this architecture fits the functions extracted from the functional analysis of the analyst's interface, the dialogue units which have been associated to these functions and also the functioning principles of the THESEUS UIMS whose main characteristics are presented in Appendix D.

In a second step, we complete our empirical validation by explaining the processing steps of some typical operations related to the K-Expert interface (such as those linked to the creation of a rule). During this stage, we are going to evoke the functioning of each architectural module concretely.

5.3.1 Compatibility between the interface design concepts and the proposed architecture

Now, each of the architecture modules is recapitulated and we say precisely what we put inside it by reference to our particular case.

5.3.1.1 The interactive objects manager

This module corresponds in fact to the THESEUS UIMS. Indeed, the objective assigned previously to this module is satisfied. As a matter of fact, the hiding of device specific Input/Output functions is supported by THESEUS seeing that like it is underlined in Appendix D.this UIMS supports a high-level of abstraction. Consequently, the application itself is not concerned by the dialogue description.

Generic interactives objects and associated generic actions are defined by THESEUS. Among them we find :

- Windows (text or graphic oriented) ;
- Icons ;
- Buttons ;
- Menu titles and menu items ;
- Graphic objects (basic ones presented in Appendix D and complex ones) ;
- Text objects (input and output fields) ;
- Dialogue boxes (not yet implemented but planned in the near future) ;
- Scrollable lists (not yet implemented but planned in the near future) ;
- Cursor (not yet implemented but planned in the near future).

The associated generic actions that may be performed by a user on these objects are presented in Section D.4. of Appendix D.

Now, we are going to show how the proceeding way of THESEUS is similar to the functioning of the interactive objects manager. In fact, THESEUS is informed of the occurrence of low-level user inputs (such as clicks on interactive objects) and then it interprets them with the help of an "event-handler" in a way that is going to be described in this part of the work.

On the other hand, when THESEUS has to display information to the user, it has to convert it from a high-level output to a lower one in such a way that it can be manipulated by an underlying window manager.

Let's see in more details the treatments operated by THESEUS on input data and then on output data. We follow the same pattern as for the architecture proposition.

◊ Taking into account of the input :

Input data : An physical user input.

Precondition :

None because THESEUS is able to accept each user input even one that does not belong to the interactive objects presented on the screen. However, the user input abilities may be precised by saying that the possible input are :

. pressing on a mouse button ;

. releasing a mouse button ;

. moving the mouse while pressing on one of its buttons ;

. keyboard input (single keystroke).

Output data :

- Updating of the screen ;

- Display of an error message ;

- Production of a sound (beep) ;

- Triggering off a function designated by a name and associated to a list of parameters. Generally, this one contains :

  . the identifier of the window concerned ;

  . the identifier of the input set concerned ;

  . the identifier of the input element concerned ;

  These identifiers are defined by THESEUS when the corresponding components are created.

  Note that the "interactive message" evoked when the architecture has been described is replaced here by a function call. Moreover, it must be underlined that the parameters transmitted by THESEUS to the upper level module (e.g. the conversation manager) do no fit exactly those described in th presentation of the architecture. In fact, the are restricted to a predefined list imposed by THESEUS. So, if a called function needs more information than that transmitted through the authorized parameters, this function itself will have to inquire the necessary information from THESEUS.

Postconditions :

Before precising postconditions, we think it can be useful to give some informations about the functioning principles that THESEUS applies in order to transform input events into associated functions. Generally speaking, we can say that the translation relies on a step by step process which can be visualized in the following way :

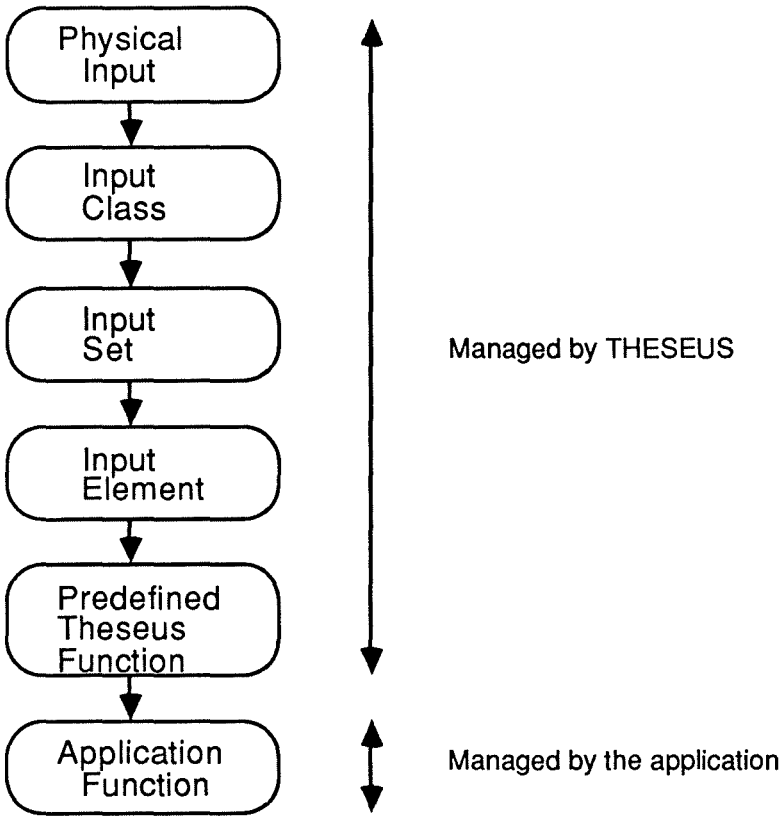


Figure 5.10 : Steps in processing user input.

Indeed, whenever an user input occurs, it is captured by the so-called "event-handler", this one tests then the authorization to accomplish the user performed action, executes treatments supported by THESEUS itself and triggers off application functions if some are defined. To be complete, it can be useful to detail each step.

"Physical input" :

Detection of a physical event performed by the user (external control).

"Input class" :

The event handler binds the physical event to an allowed input class. The authorized ones are :

. menu selection (selection of one menu item in a restricted number of alternatives) ;

. icon selection ;

. object identification (picking of an object visible at the screen within a window) ;

. position area (entry of a position in world coordinates within predefined areas.) ;

. keyboard input (pressing on a key) ;

. object dragging (mowing of an object within a window) ;

. icon dragging (moving of an icon around the screen) ;

. window related input (size, move, scroll, close, help or undo function).

"Input set" :

After the identification of the input class, the event handler determines which corresponding input set is concerned. It can be underlined that each input class is divided into several input sets. The correspondences are the following one :

Input class	<------->	Input sets
- Menus selection		-Menus
- Icon selection		-Icons sets
- Object identification		- A set of objects consisting of all existing modes and another set consisting of all existing edges

- Position Area	-Sets defining various input areas (example : one set may be defined for each window)
- Keyboard Input	- Sets of various keys (example : the set oflower case characters)
- Object dragging	- Sets of all the objects that can be moved

"Input element" :

Knowing the input set, the event handler can then go into more details by identifying the concerned element of the retained input set. For example, the elements of a "menu" input set are menu items. For each of them, the following informations are accessible to the event-handler :

. name of the item ;

. number of the item in the set ;

. item state (enabled or disabled) ;

. special item label (e.g. check mark) ;

. name of the application to call when the item is selected (if there is any).

"Predefined THESEUS function" :

At this step of the process, THESEUS should have identified precisely the concerned element. So, if some treatment may be performed by itself without involving the application, it performs it before passing the deal to an application function if one is associated to the concerned element. Actions that THESEUS may perform independently of the application have already been enumerated at the beginning of this Section. As an example, we can think to the following case : the action of

the user consists of a click on the close box of a window. When this event is detected, the event handler performs all the necessary treatments to identify the corresponding input element. When this is done, THESEUS "closes" the window. It means that it erases it from the screen and reactivates the last activated one. Then, if there is an application function associated to it, it triggers it off. Similarly, when THESEUS detects a keyboard input, it verifies the permissibility of this event and calls an application function, if necessary, afterwards.

"Application function" :

Generally, THESEUS calls such a function by passing a parameters list which has been defined in the output clause of this point. This corresponds to the sending of an interactive message to the conversation manager.

Now, let's come back to the expression of postcondition.

- The module produces a sound if the event perceived can not be identified as an existing input class ;

- The module produces an error message if a function accomplished by THESEUS itself (such as a syntactical control related to a pattern of characters associated to an input text field) highlights a problem that the user has to manage ;

- The module updates the screen if the event leads to the accomplishment of a function managed by THESEUS autonomously and that affects the external shape of some objects (such as the dragging of an icon, the resizing or closing of a window) ;

- The module triggers off an upper-level function on the provided list of parameters (see "output") if the identified "input element" has been linked to a particular function that has to be performed in response to the event. However, the

execution of this function is beyond the THESEUS scope, it is under the responsibility of our "conversation manager" as it will be shown in the next section.

◊ Taking into account of the output :

Input data : Triggering off a THESEUS function by giving the following elements to this module :

- The function name ;

- The parameters corresponding to this function.

In fact, this corresponds to a part of the interactive message considered in the architecture proposal. The conversation manager may give the lead to THESEUS by calling function.

What corresponds exactly to the interactive message defined previously in the input clause of the "output translation function" is in fact a sequence of THESEUS functions triggered off each in its turn. For example, to display a complex interactive object, a function of the "conversation manager" must create each of the components of this object by calling several THESEUS function.

Precondition :

The given function name corresponds to an existing THESEUS function. The given parameters are the right ones and they contain an authorized value. To be more precise, we can say that the general classes of THESEUS functions are the following ones :

. creation (of instantiations of a generic interactive object);

. updating (of instantiations of interactive objects by changing their attributes that can be modified dynamically -see Section 5.3.) ;

. deletion (of an instantiation of such an object) ;

. inquiring of information about an instantiation (for example the list of the sons of a complex object) ;

. moving (of an instantiation of an object) ;

. starting and stopping the event-handler (e.g. triggering off or concluding the THESEUS work) ;

. display of a dialogue box (containing a particular message appearing as a parameter of this function).

The reader who would like to know precisely the function names and their associated parameters may consult [THESEUS 4].

Output data : The triggered off function is performed.

Postconditions :

The result of the function execution is in conformity with its particular postcondition. Generally speaking, it can be said that the effect of an execution may have a direct effect on the screen contents or may return information to the calling upper-level function via parameters settings.

5.3.1.2 The conversation manager

In our concrete case, this module corresponds to the main program of the interface for K-Expert. It is in fact a source code program written in C. Indeed, at the time being, the way to use THESEUS to implement a man-machine interaction may be described in the following manner :

- Generally speaking, a so-called "interface program" must signal to THESEUS in which way it has to react to which actions. It means that this main program must create and initialize the data structures that must be displayed as interactive objects on the screen. For example, it has to define windows by specifying the desired components (such as the "close box", the "size box", the "title", ...) and the functions to associate to these components if a special treatment is required. The

source code corresponding to the associated functions has also to be written in this program. This code may rely on functions managed by THESEUS and which belong to the categories defined in the taking into account the output in Section 5.3.1.1. In order to trigger off the THESEUS event-handler so that it can perceive user physical inputs, the main interface program is also responsible for the starting and the stopping of the event-handler. This is for the present. In the near future, this programming could be replaced, up to a certain extent, by a dialogue generator which is evoked in Appendix D Then the necessary C-code (contents of the conversation manager) could be automatically produced. What should also be kept in mind is that it is the main program, we are speaking about, which implements the dynamic of the dialogue through the definition of the functions that it associates to each possible event. The contents of this module is built on the basis of the informations presented in Section 4.13.3. ...

From a general point of view, it can be said that this module fits well into the characteristics proposed in its abstract description.

From a particular point of view, we can say that this module

- triggers off THESEUS functions in order to perform its own functions.
- triggers off proper application functions in the same aim. The specification of these application functions may be found in Section 4.6. .
- receives complementary informations (through a parameters setting) from THESEUS (by executing an "inquire THESEUS function") or from an application function it triggers off.
- is activated by THESEUS or by application functions which want to trigger off some of the functions it possesses.

5.3.1.3 The interactive application manager

In our concrete case, this module is a program written in C which plays the role of correspondence table between the application functions necessary by the conversation manager module and those offered by the application functions module and conversely. Moreover, this module enables external programs to have access to functionalities of the interactive application. So, this module is an interface providing access to the library of functions of the application functions module and the library of functions of conversation manager module. The available application functions are described in Sections 4.6. and 4.7. .

As for the conversation manager functions, they can be deduced from Section 4.8. and 4.11. . So, the contents of this module has to be written by the interface developer on one side and by the application functions developer on the other side in order to implement the transparence between the interface and the application.

This module decides which control must be the leader. So, in our case, the module disposes of a procedure written in C which calls the main function of the conversation manager module (e.g. the function starting the event-handler of THESEUS).

5.3.1.4 The application functions

In our particular case, this library of functions is a black-box. This module is under the responsibility of the developers of the application functions module. Thus, it is not relevant for the study of the interfacing of an interactive application. The contents of this module can be built starting from the functions specified in the Section 4.6. .

Let's now conclude our empirical architecture validation by the presentation of processing steps associated to a basic functionality of the proposed interface.

5.3.2 Processing steps of a typical functionality of the K-Expert interface

The aim of this section is to show that all the steps occurring during a using session of the interface that has been proposed for K-Expert may be created by well-defined modules of the architecture we have conceived.

To fulfil this global objective, let's refer to a typical use of the interface. Among typical and most frequent operations performed thanks to the interface, there is the creation of a rule.

In order to be complete, we start from the launching of the K-Expert interface but we consider that all the necessary preliminary actions have been performed. These actions are the selection of an expert system and of a knowledge module.

To preserve the understandability of this example, we assume that no wrong manipulation is performed and consequently, no error-treatment has to be envisaged. However, we consider some typical actions that can be performed at any time, whenever one has loaded the interface, such as resize a window.

Notice that the actions performed by the analyst are put into signs like these : < >. Let's now begin the example.

< Type "K-Expert" to launch the interface >

Interactive application manager :

- Activates the conversation manager by triggering off the program corresponding to the K-Expert interface.

Conversation manager :

- Begins the execution of the code instructions. It calls a THESEUS function in order to initialize the whole THESEUS work.

Interactive objects manager :

- Initializes its work.

Conversation manager :

- Continues to execute its instructions. Concretely, it transmits parameters to the interactive objects manager in order to create the K-Expert window.

Interactive objects manager :

- Updates its internal structures (input sets, input elements...) according to the received parameters.

Conversation manager :

- Starts the event-handler.

Interactive objects manager :

- Displays the K-Expert window.

(...etc.)

< Click on the "Editor" icon of the K-Expert window >

Interactive objects manager :

- Perceives the event, identifies it and finds the corresponding function name by applying the method explained in the Section 5.3.1.1. called "Taking into account of the input" ;

- Triggers off the corresponding function of the conversation manager.

Conversation manager :

- Gives parameters to the interactive objects manager in order to create and display the "Editor" pop-up menu.

Interactive objects manager :

- Updates its internal data structures ;
- Displays the "Editor" pop-up menu.

< Click on the "Rules Editor" item in the "Editor" pop-up >

Interactive objects manager :

- Perceives, identifies the event and finds the corresponding function name ;
- Triggers off the corresponding function of the conversation manager.

Conversation manager : (1)

- Requests the interactive application manager to obtain the first rule of the considered knowledge base.

Interactive application manager : (2)

- Triggers off the chosen function of the application functions in order to provide the result awaited by the conversation manager.

Application functions : (3)

- Fetches the first rule from the knowledge module ;
- Returns it to the interactive application manager.

Interactive application manager : (4)

- Transmits the received rule to the conversation manager.

Conversation manager : (5)

- Gives parameters to the interactive objects manager in order to create and to display the "Rules Editor" window with the received first rule as contents.

Interactive objects manager : (6)

- Updates its internal structures ;
- Displays the "Rules Editor" window.

(...etc.)

< Click on the "next page" icon >

Interactive objects manager :

- Perceives, identifies the event and finds the corresponding function name ;
- Triggers off the corresponding function of the conversation manager.

Conversation manager :

- Requests the second rule of the knowledge base via a process similar to that described in order to obtain the first one. This process is not described again here. So, we assume here the realization of the step (1) to (4) ;
- Gives parameters to the interactive objects manager in order it updates the Rule Editor window with the received second rule.

Interactive objects manager :

- Updates its internal structures ;
- Displays the "refreshed" Rule Editor window.

< Click on the size box of the Rule Editor window >

Interactive objects manager :

- Perceives, identifies the event. As there is no explicit function name associated to this event, it treats it by itself. As a result, the window is resized after an updating of the corresponding internal structures.

< Click on the "New" button of the Rule Editor window >

Interactive objects manager :

- Perceives, identifies the event and finds the corresponding function name ;
- Triggers off the corresponding function of the conversation manager.

Conversation manager :

- Create an "add new rule" functional message ;
- Requests the interactive objects manager to display an empty mask in the Rule Editor window by giving it the necessary parameters.

Interactive objects manager :

- Updates its internal structures ;
- Displays an empty mask.

< Entry of a rule name in the corresponding field of the Rule Editor window >

Interactive objects manager :

- Perceives, identifies the event and finds the associated function name ;
- During the identification process as THESEUS discovers that a pattern of authorized values for each character has been attached to this field, it performs a syntactical control ;
- If there is a problem, it produces of its own an error message box to draw the analyst's attention. Then, it waits for a new event from the analyst (in fact a click on the "OK" button of this box) ;
- If there is no problem, THESEUS triggers off the corresponding function of the conversation manager.

Conversation manager :

- Requests the interactive application manager to verify if the typed rule name already exists in the considered knowledge module.

Interactive application manager :

- Triggers off one of the functions of the Application Functions in order to provide the result awaited by the conversation manager.

Application Functions :

- Consults the database to obtain the desired answer ;
- Returns the answer to the interactive application manager.

Interactive application manager :

- Transmits the received answer to the conversation manager.

Conversation manager :

- If the rule name already exists, it gives parameters to the interactive objects manager in order to display an adequate error message ;
- Else (e.g. the rule is a new one), it adds the rule to the "add-new-rule" functional message.

< Filling of the other fields composing the Rule Editor mask >

The treatment of these fields is similar to this that has just been presented.

< Click on the "Accept" button of the Rule Editor window >

Interactive objects manager :

- Perceives, identifies the event and finds the associated function name ;
- Triggers off the corresponding function of the conversation manager.

Conversation manager :

- Verifies if all the necessary information have been put into the functional message "Add-new-rule" ;

- If the functional message contents is not satisfying, the conversation manager gives parameters to the interactive objects manager in order to display an adequate error message ;

- Else (e.g. the message contents is all right), the conversation manager requests the interactive application manager in order to add the new rule for which the information has been seized from the analyst in the database. It gives the built functional message to this module as a parameter.

The same type of treatments occurs up to the moment when THESEUS detects a "quit" event. Then, the event handler is stopped and the dialogue is ended.

5.4 CRITICISM OF THE PROPOSED ARCHITECTURE

In this chapter, we have proposed an architecture that is flexible enough to be applied to every kind of interactive application. Indeed, it can be instantiated with interactive applications that have an internal control with hooks to mixed control but also with interactive applications in which the control is external and offers hooks to mixed control.

Moreover, the proposed architecture strives for the modularity principle. Indeed, we have regrouped concepts of the same nature in each module. The objective of each of them is well-defined and the interrelations between them are well-known and punctual.

The principle of independence between the application and the corresponding dialogue is respected. It makes possible to perform a separate work on these two parts as well during the conception or execution steps as during the maintenance stage. By this way, the dialogue conception can be realized by persons who are specialists in the domain of ergonomy while the application functions may be specified by experts in computer science.

During the implementation, the correspondence between the two independent constitutive parts is defined thanks to the specification of the interactive application manager.

During the execution stage, the interactive application manager manages also all the communications which are necessary if the interface runs on another computer than that used for the performing of the application.

The principle of independence is also respected during the maintenance step. Indeed, the implementor can change some of the dialogue features without having to modify the application part. Of course, this constatation is only valid while the implementor does not change the specification of the function He modifies.

The proposed architecture is also complementary to the elements described during the specification of the considered interactive application (See Chapter 4.). As a matter of fact, all the functions derived from the retained basic functionalities may be regrouped into the application functions module. In other respects, all the interactive objects and their chaining can be put into the conversation manager module.

To conclude, let's note that we have validated this architecture by instantiating it to the THESEUS UIMS and to a typical script of use of our particular interactive application. Nonetheless, in order to validate our architecture completely, a complementary step is necessary. It consists of the implementation of the interfaced interactive application but this point is beyond the scope of this work.

As conclusion, we describe the aspects which have been tackled in this study. Then, we place our contribution to the carried out work. Finally, we present some future prospects and also the new topics of development and research that can be considered.

1. TACKLED ASPECTS

We have presented theoretical principles and more empirical ones that should always inspire man-machine interface designers whatever interactive application they consider. Then, keeping in mind these general pre-requisites, we have gone deeply into our thought in order to adapt it to a particular interactive application such as the building and consultation of an instantiation of an expert system shell.

In this perspective, we have taken into account two particular points of view. First, the person for who we had to propose an interface (e.g. an expert system analyst) and afterwards, the person who is responsible for the implementation of the suggested interface.

To be more precise, let's say that in a first time we have determined basic features of the interface to design by thinking about the profile and also about the task of the considered analyst. Then, to facilitate our design process, we have looked at already existing expert system shells interfaces. We have had a practical approach of them (e.g. personal experience) and we have criticized them at the light of the previously evoked theoretical principles.

The next step consisted of the functional analysis and of the specification of our interactive application. At this stage of the work, we were able to propose a visualization of the conceived analyst's interface independently of implementation concerns.

Then, we have tackled the problems linked to a possible implementation of our analyst's interface. In order to solve them, we have proposed to retain a modular architecture in which the dialogue is separated form the application functions.

Finally, we have introduced a so-called empirical validation of the architecture we have built in order to cope with our impossibility to realize the implementation of the interface.

2. CONTRIBUTIONS TO THE WORK

The theoretical principles follow form studies led by various kinds of people (computer scientists, psychologists, ergonomists...etc). Indeed, we have been careful to reflect the fact that the interface designing must be submitted to a pluridisciplinary approach in order to face the diversity of the human intervenings. The general approach of the analyst's task and profile is also inspired by already carried out researches.

The empirical study of existing interfaces and of the expert system shell to interface (e.g. K-Expert), on their side, result from our practical experience with them. This experience has been acquired notably during our training period in Germany.

As for the specification of the considered interactive application and the proposal of an implementation architecture, we have been inspired mainly by researches led at the "Institut d'Informatique" at Namur. Indeed, we have adapted current results of these researches freely in order to fit the case in which we took interest.

3. TOPICS OF DEVELOPMENT AND RESEARCH

Generally speaking, it can be underlined that many parts of the work presented here may be submitted to further researches. For example, as it often appears through the literature, the domain of theories concerning the man-machine communication is in full development at the time being.

Pluridisciplinary studies in this domain should be pursued in such a way that it would be possible to modelize the way humans interact with computers and to visualize their task as accurately as possible. In this perspective, it appears also that one should concentrate efforts on the determination of human mind abilities and limitations in order to adapt computers and also programs to them (and not the contrary).

In a word, let's say that after the building of solid foundations for the designing of ergonomic physical tools (such as keyboards, mouses and so on), one must be aware that such an effort must also be consented in order to design what becomes known as ergonomic softwares.

As for our personal contribution, it seems clear to us that it consists of the first steps of an interface designing process. As a result, our work should be submitted to various refinements so that to an implementation. In particular, let's underline that the specification process we have proposed and illustrated by the study of some functionalities should be realized in a systematic way for all the retained functionalities.

The analyst's interface proposal, on its side, results from our readings and personal experience with existing expert system shells interfaces. Of course, we have tried to approach the analyst's task and profile in order to design our interface on solid bases.However, we are not analysts and we have a certain computer science background at our disposal. Consequently, it should be more than necessary to take our proposal as a preliminary study which has to be submitted to a prototyping step in order to adapt it to real analyst's needs and abilities. For example, we should not forget that analysts do not always dispose of extended computer knowledges.

Another domain of research concerns the implementation of software environments able to support the designing process.

The architecture we have proposed should be valid for every interactive application. We have validated it by showing that it could be possible to affect a consistent contents to all its modules if we decide to implement our interface proposal with the THESEUS UIMS.

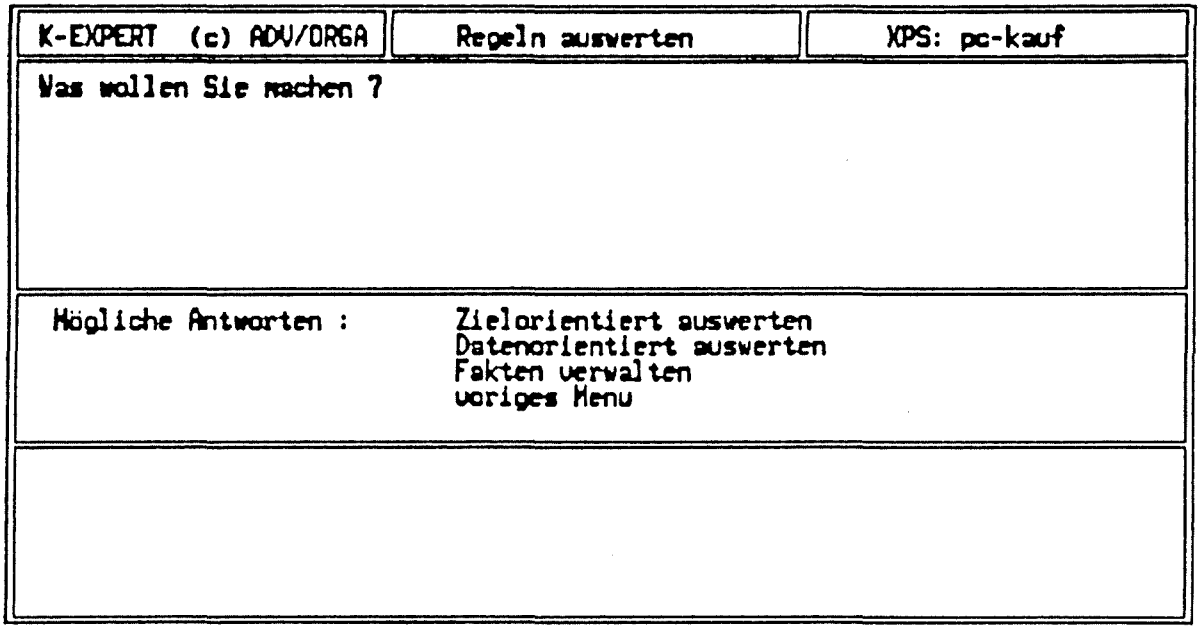
Obviously, the best validation for such an architecture would consist of an implementation by programmers. During this process, it should be possible to determine if it is really possible and suitable to consider the modules we have retained and to inter-link them as we have done it.

As a general concluding remark, let's say that what follows from our work and what should be kept in mind while interfacing man-machine interactions is that the process of designing interfaces relies on many tradeoffs. Consequently, interface design should be seen as an Art requiring the intervention of numerous kinds of people rather than as a systematic and predefined process which could be defined once and for all.

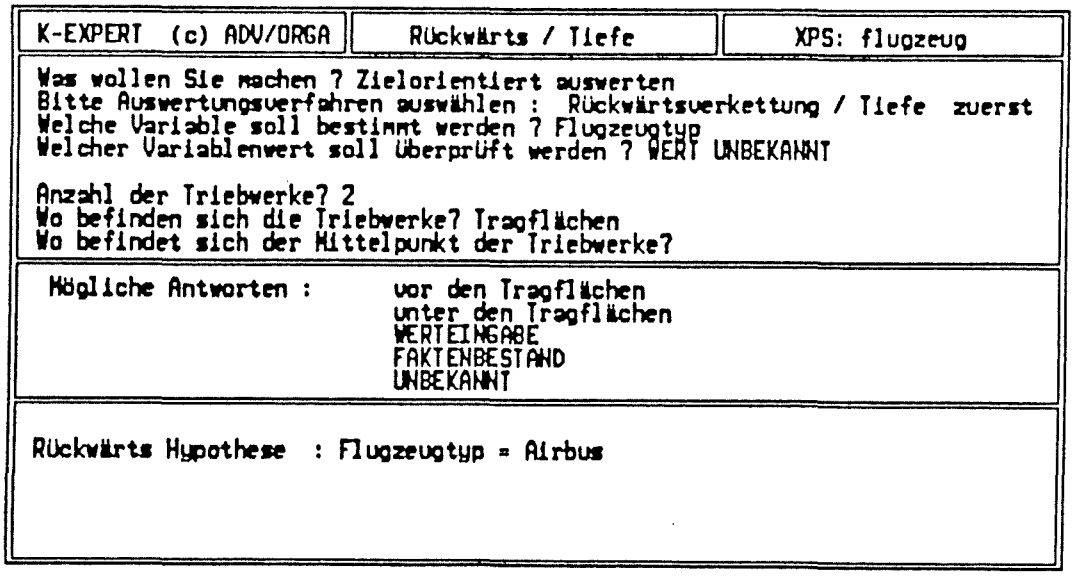
APPENDIX A :

K-EXPERT ILLUSTRATION

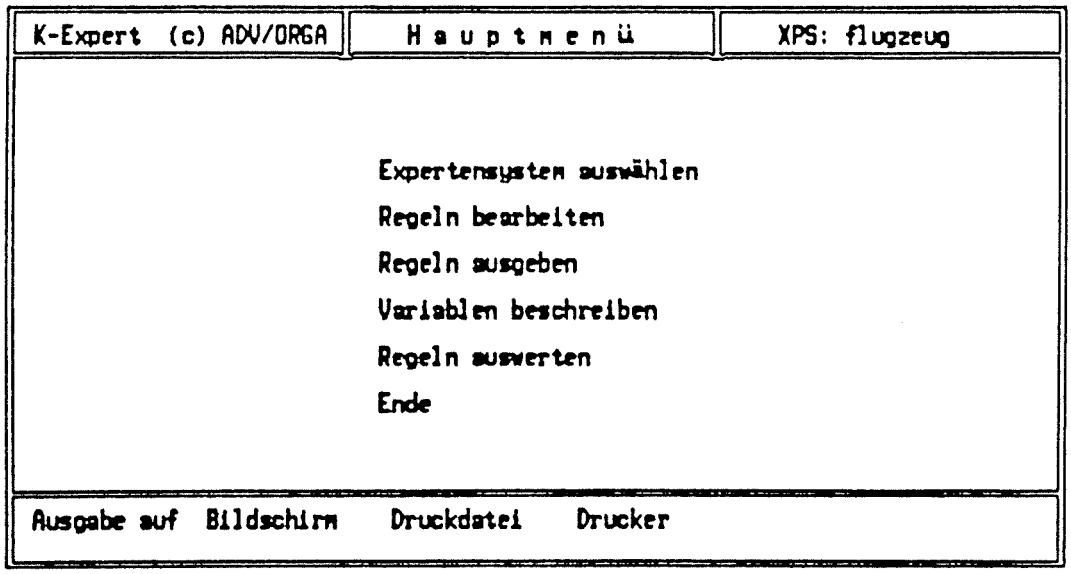
◊ Figure A.1 : Initial Run-time environment.



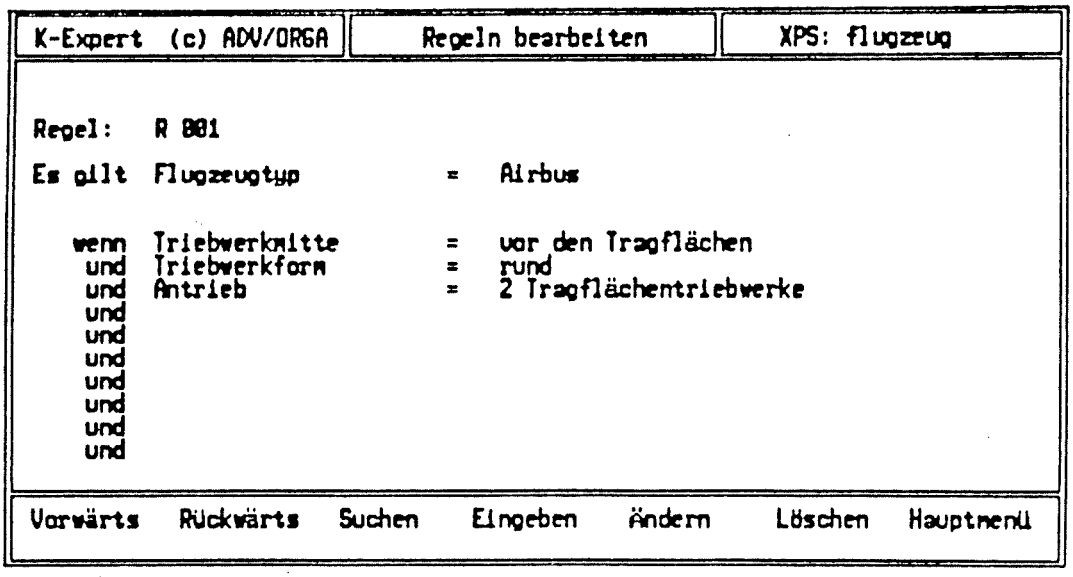
◊ Figure A.2 : Run-time environment during a consultation.



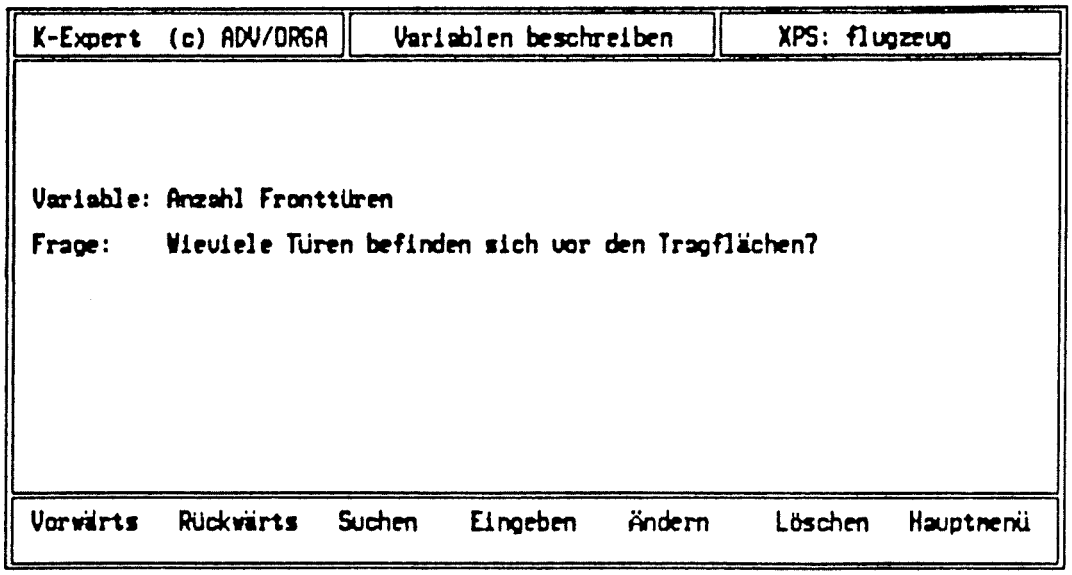
◊ Figure A.3 : Initial knowledge engineering environment.



◊ Figure A.4 : Rules Editor.



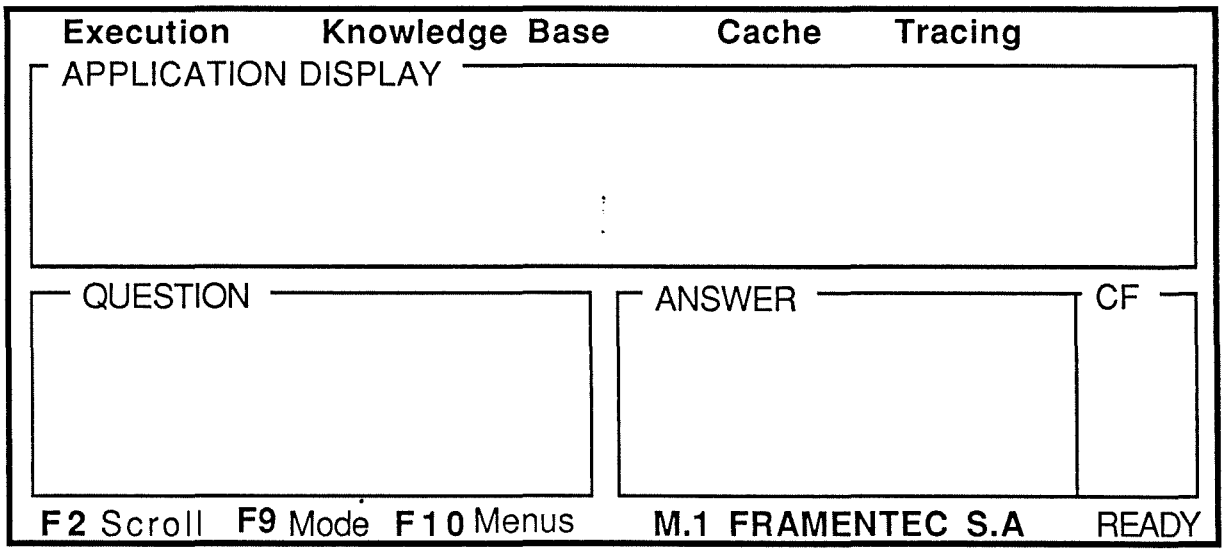
◊ **Figure A.5 : Variable Editor.**



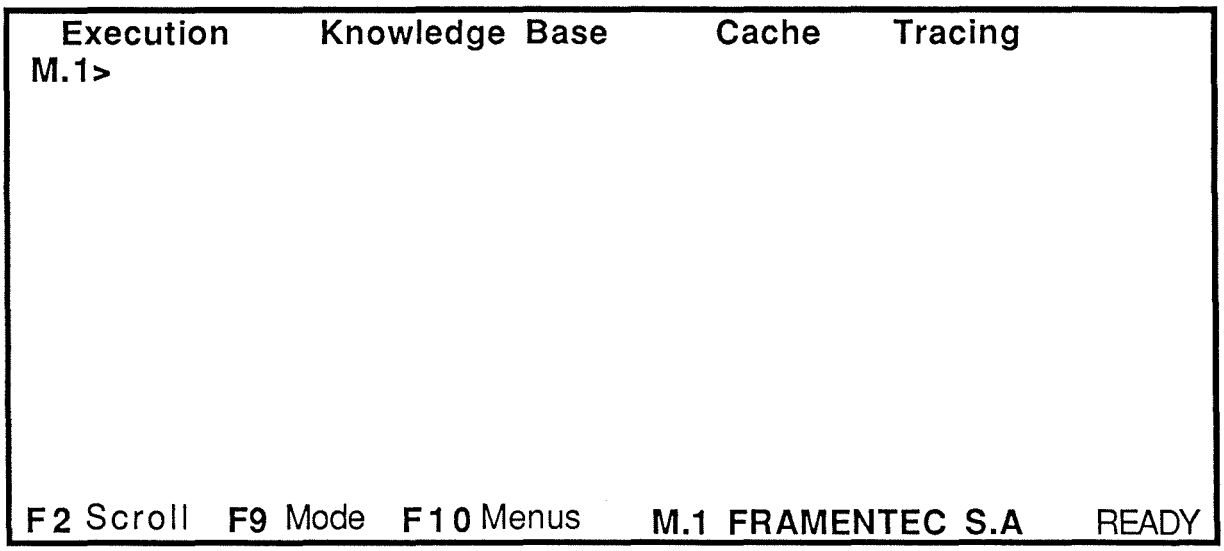
APPENDIX B :

M.1 ILLUSTRATION

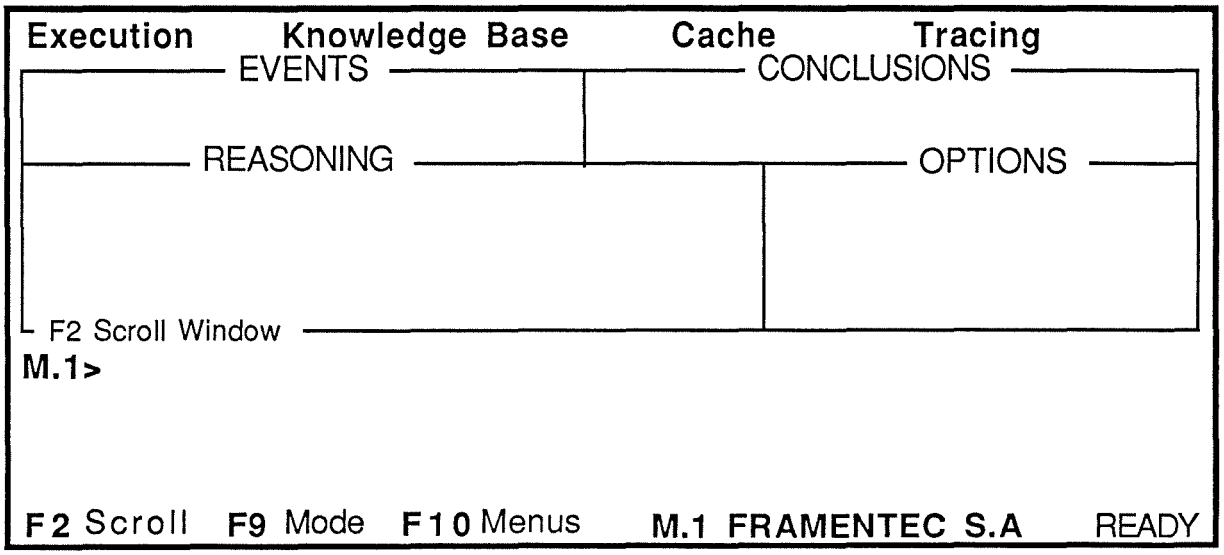
◊ **Figure B.1 : M.1 delivery environment.**



◊ **Figure B.2 : M.1 development environment.**



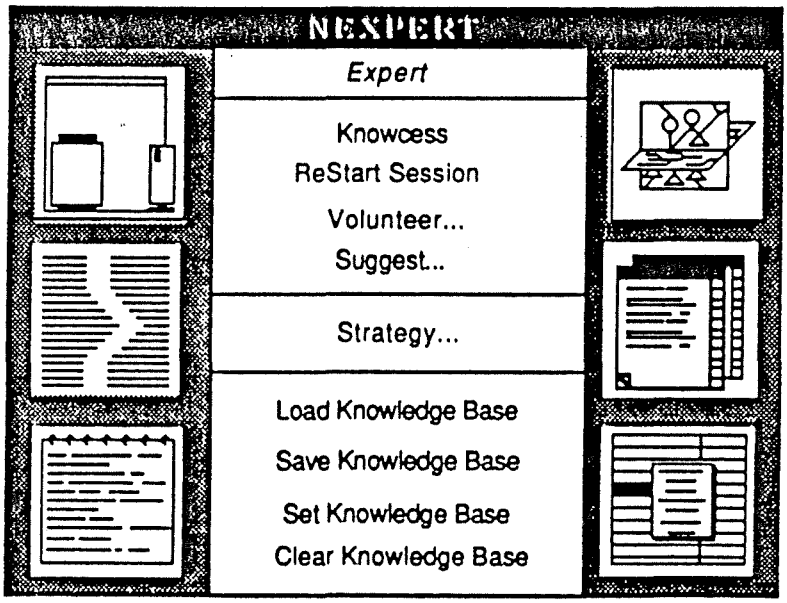
◊ **Figure B.3 : M.1 development environment with the panel option "on".**



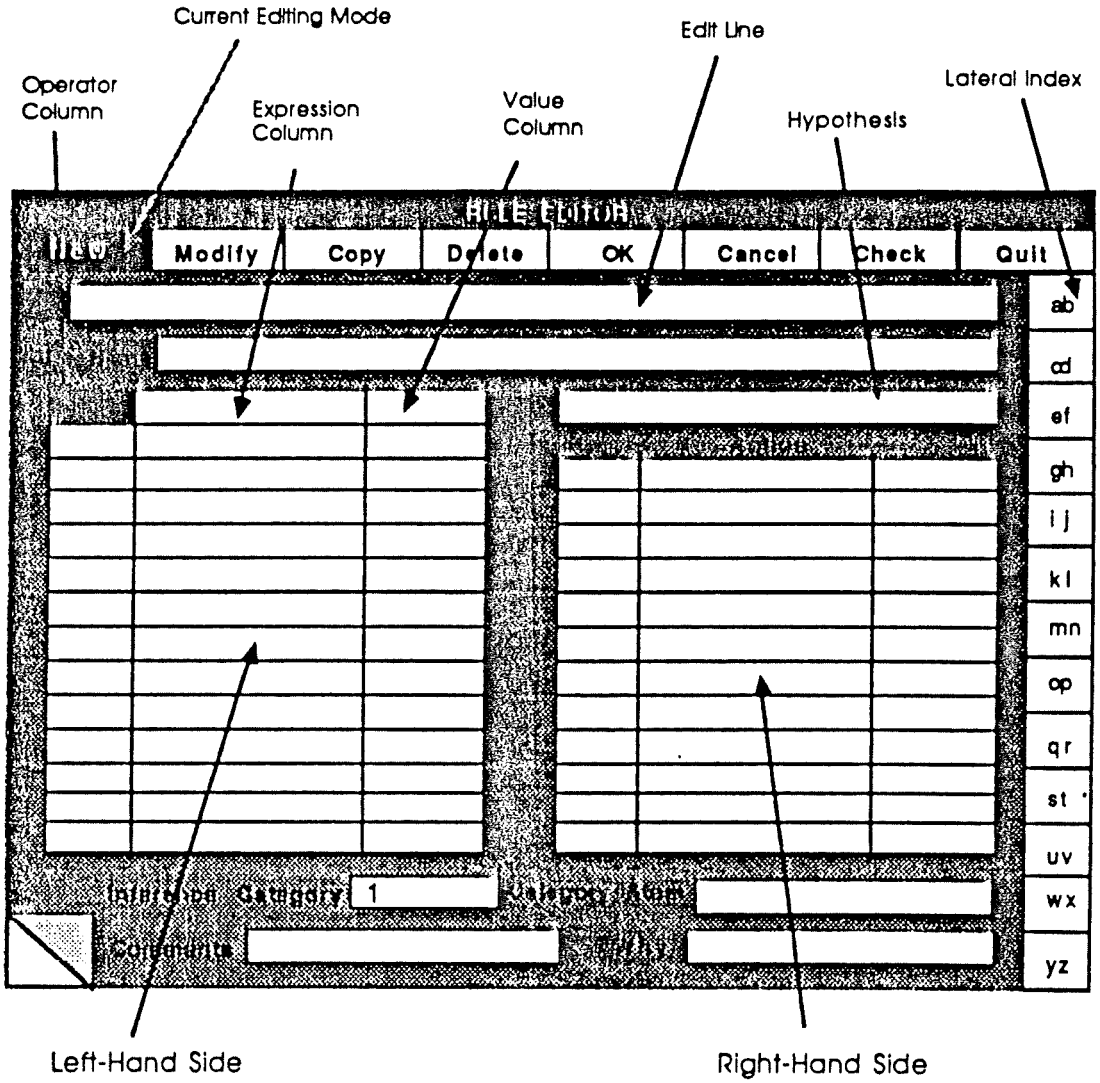
APPENDIX C :

NEXPERT ILLUSTRATION

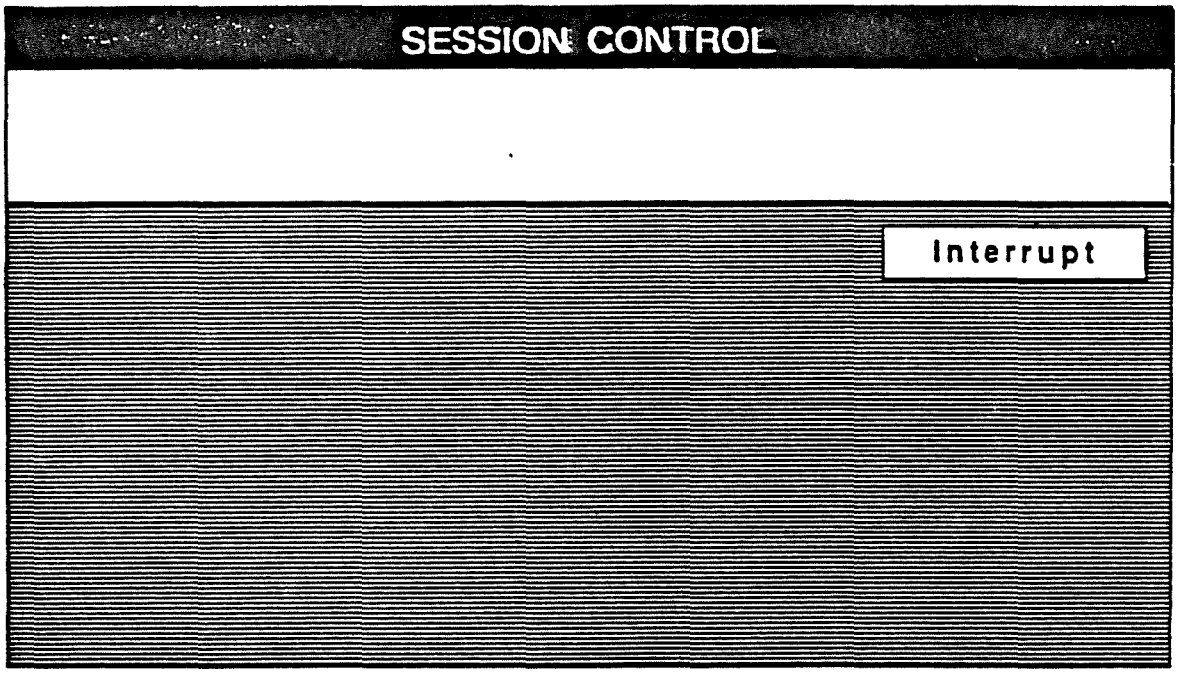
◊ **Figure C.1 : Initial window.**



◊ **Figure C.2 : Rules Editor window.**



◊ **Figure C.3 : Session control window.**



APPENDIX D :

MAIN CHARACTERISTICS OF THE THESEUS UIMS

D.1. INTRODUCTION

From a general point of view, THESEUS can be defined as a User Interface Management System (UIMS) allowing to design and control graphical user interfaces in a multi-window environment. It supports state-of-the-art interaction techniques such as :

- Multi-windowing for parallel work in various contexts ;
- Use of graphics (instead of exclusive use of text) for condensed and visual display of informations ;
- Comfortable mechanisms like menus, icons (and so on) for user input ;
- Object oriented output adapted to the user level. Among the object oriented concepts, there are fixed basic objects classes :
  - . "polyline objects", graphics primitives consisting of a straight line which may or may not have a direction such as a circle arc ;
  - . "area objects bound by edges" like boxes, triangles...etc ;
  - . "graphics texts objects" containing alphanumeric strings ;
  - . "master objects" consisting of arrays of pixels with fixed physical size ;

It is also possible to define complex objects by recourse to structuring mechanisms applied to basic objects. These graphics objects are controlled via graphics attributes such as geometrical attributes, size and dynamical attributes (as edges type, edges visibility, display of a graphics text...etc). Moreover, an inheritance mechanism is used in order to bind complex object attributes to son objects.

- Event-driven input processing to implement user-controlled dialogue techniques. It means that the description of a dialogue is based on an event-model in which user interactions with physical devices are put into correspondence with a predefined set of events which are handled by a so-called "event handler".

The main application areas for THESEUS are software development tools and software engineering environments. In a word, these environments may be characterized by the fact that their underlying philosophy of the user interface is based upon a dialogue possessing a type and a contents determined by the user and not by some fixed sequences built into the application itself.

This UIMS, whose name means "THE Software Engineering USer interface management system" is developed by the "Zentrum für Graphische Datenverarbeitung" in Darmstadt. It is now in full revision in order to fit more and more to its predefined aims. These ones may be summarized in the following way :

- THESEUS must ensure the separation between tasks dealing with man-machine communication and the application itself by putting them into an independent component supporting flexible communication mechanisms. This separation is particularly significant for open systems such as software engineering environments because these ones have to integrate a lot of software development tools.

- It supports a high level of abstraction. In this way, application functions are released of all tasks dealing with the dialogue interface program and, moreover the programmer must not think about the realization of specific interaction techniques like menu and icon selection, dragging or object selection.

- As far as possible dialogues are executed locally without involving the application program. This is perhaps one of the most interesting characteristics of THESEUS for an interface programmer. For example, all window operations such as resize and close are handled by THESEUS itself autonomously when most of the existing window managers involve the intervention of application functions in charge of the treatment of such operations. In other words, the window

management in THESEUS hides all the problems normally related to multi-windowing from the application program. Another example of this independence of the application is that a large output buffer of variable length is connected to every alphanumeric window independently of the actual size of the screen which is also handled by THESEUS automatically. Only a part of this buffer is visible in a window at a time but a scrolling ability supports the shifting of the visible area without requiring any intervention of the application. Moreover, it can also be mentioned that the dragging of graphics objects is also entirely under the THESEUS responsibility. This feature is envisageable because THESEUS is at an upper level than a classic window manager. Indeed, we can visualize the THESEUS situation as follows :

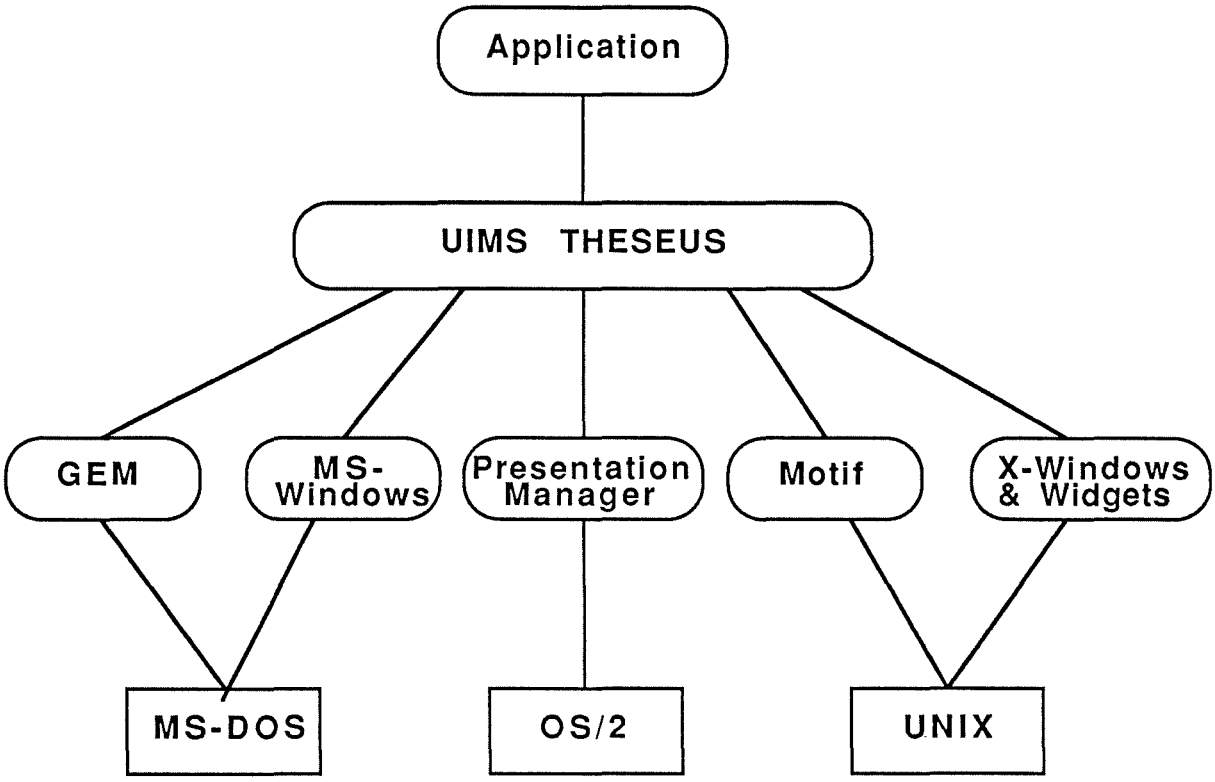


Figure D.1. : THESEUS development directions.

- With regard to the control, it can be said that the flexibility already evoked in the communication interface <---> application is reinforced by the fact that the control implemented is an alternating one. Indeed, an application function requiring information from the user or having to

display it calls THESEUS, so we can speak of internal control. On the other hand, the application program is called by THESEUS in order to support input events by providing corresponding functions. This may be related to external control. This fits dynamic dialogues particularly well because there, the flow of the control has to be changeable at the run-time. We have already said that THESEUS is event-driven. So, according to this feature, its control model may be represented in the following way.

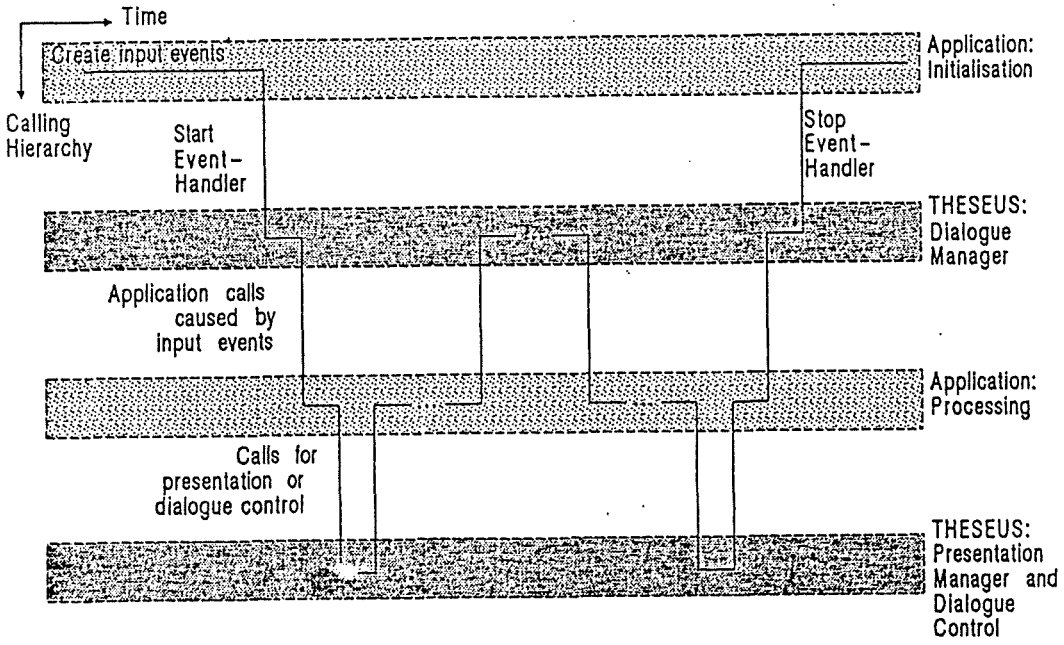


Figure D.2. : THESEUS Control Model.

A final remark about control is that THESEUS implements the philosophy of dialogue dominance. It means that all the changes visible on the screen are triggered off by the user only ; indeed, functions like size or scroll a window are only offered to the user and not to the application program. We can also remark that, at the time being, it is not possible to implement interactive application in which, during the dialogue course, an application function requires information. In a word, it can be said that it is now impossible to start a dialogue inside a dialogue previously begun. This is annoying for us because it means that the starting of an expert system consultation from the dialogue implemented through the analyst's interface is not envisageable.

However, this is only a temporary limitation. Indeed, in the near future, it should be possible to implement a so-called "temporary event-handler" which will only be active during the seizure of the necessary informations.

- Moreover, THESEUS provides a consistent user interface for all the software tools of an integrated system. So, there is a uniform model which combines all the different techniques of dialogue programming, graphics programming and window management. The consistency is reinforced by the fact that THESEUS respects the Common User Access principles [IBM 87]. So, an application interfaced with THESEUS should be consistent with other ones which embody the rules highlighted by the Common User Access.

- Finally, thanks to THESEUS, it must be possible to develop user interfaces by an incremental dialogue specification. Indeed, an interactive procedure can be followed during interface design. So, one can start from a prototype of the end-product to implement. During every dialogue step, the only thing to do is to describe the difference from the preceding steps. By this way, the complexity of dialogue description is alleviated and the ability of dynamic modifications increases the flexibility of the dialogue specification. Indeed, at any time, it is possible to add or suppress events kept in mind by THESEUS.

D.2. THE THESEUS LAYER MODEL

Now, more details may be furnished about the architecture of THESEUS. We present here a logical view onto the THESEUS system as it is seen by the system developer. In fact, the task of transforming device-specific functionality to application-oriented semantic and conversely is realized by three layers. This model may be visualized thanks to Figure D.3.

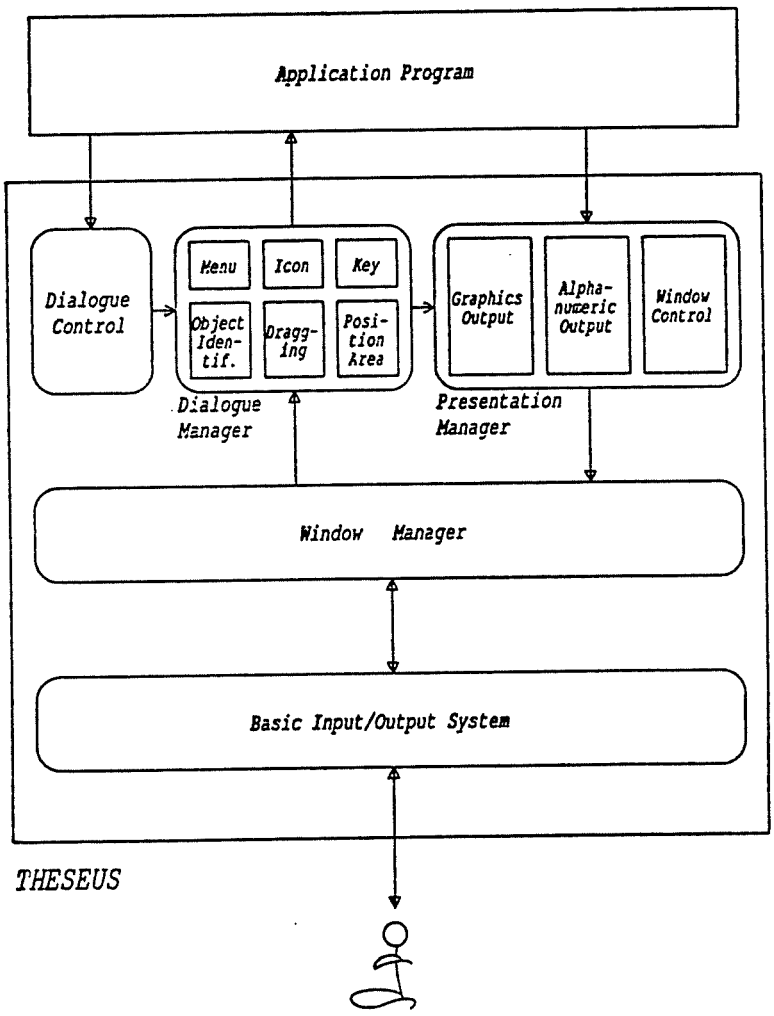


Figure D.3 : THESEUS System Architecture.

The **Basic I/O system**, the first level, maps device capabilities to a device independent level.

The **Window manager**, the second level, repairs overlapping screen areas, maps user input to input classes recognized by THESEUS and provides output primitives (such as polyline) into visible or hidden windows including transformations from window to screen coordinates.

The third and last one is composed of the following components :

- The **Presentation manager** which is busy with the display of all the informations on the screen. This functionality is

used by the application program in order to open graphics or text windows and to draw and manipulate objects within the window. The data structures of windows and their components are stored as logical information. Consequently, it is possible to perform screen repairing and updating without the intervention of the application.

- The **Dialogue manager** maps user input to logical application oriented input sets. This module works in two directions. Indeed, if the input corresponds to an application function, the application program is informed by the dialogue manager. On the other hand, using features of the presentation manager, this manager performs feedback to user inputs.

- The **Dialogue control** provides abilities to specify dialogue units and sequences by creating input sets and their elements which, for example, represent menus...etc. The information contained in this module is used by the dialogue manager to prompt, echo or reject user input.

At this level, we do not go deeper into details about the functioning process of THESEUS, it is done in the Section 5.3.

D.3. CONCLUDING REMARKS ABOUT THESEUS

Now, from a practical point of view, it can be said that the three main tasks assigned to THESEUS are :

- To ensure the availability and the management of a so-called virtual device surface (e.g. a world coordinates system for graphics windows or a coordinates system of rows and columns for text windows). It could be underlined that the information which is actually displayed may be only a part of this surface ;

- To realize local interactions (e.g. to perform actions associated to the contents of a window) ;

- To map input events corresponding to user actions with window contents and if necessary, to transmit these events to an application function which has been linked to this window.

This general presentation of THESEUS may be concluded by some remarks about the way to implement application user interfaces by using this UIMS.

At the time being, the interface programmer, refering himself to generic interactive objects and their properties which are offered by THESEUS, has to write an interface program. This one contains the instantiation of necessary generic interactive objects, associates functions to each possible event and starts (and stops) the event-handler.

However, this is a really tedious task. So, in the near future, interface programmers should have access to a so-called dialogue generator. This generator should enable them to instantiate interactively generic objects interactively. For example, it would be possible to instantiate a window by clicking on icons representing all its components in order to affect the desired ones to the instantiation.

Moreover, there would be the ability to write a function in correspondence to an event.This can be illustrated in the following manner. After selecting a button interactive object, and after having positioned it at the right place, the programmer could click on it in order to open a dialogue box in which to type the text of a function to be launched when the button will be clicked on during the run-time.

However, in a first time, the ability to directly write functions in correspondence with events will be restricted to low-level functionalities (such as the display of a check mark in front of a selected menu-item). Indeed, for functions which imply the intervention of the application, only the function name will be given. The contents of the desired function written in C language will have to appear in another place. At the end of the use of the generator, an ASCII resource file should be created. This one should be integrated with the C functions written in a common editor in order to create a single file understandable by THESEUS.

D.4. GENERIC ACTIONS ON GENERIC INTERACTIVE OBJECTS

◊ Actions on a window :

Managed by THESEUS autonomously :

- Click on a size box (to resize the window) ;
- Click on a move box (to move the window) ;
- Click on a scroll bar (to scroll the window contents) ;
- Click on a already opened window (to activate it and to give it the status of a so-called "listener window") ;
- Click on a iconization box (to iconize the window) ;
- Click on a growing box (to extend the window to a screen size) ;
- Click on a zoom box (to change the size of the objects contained in the window) ;

The three last options will be implemented in the near future.

Managed by particular functions :

For all the actions of this kind, THESEUS performs default treatments (if there is one) and then calls the attached function (if there is one too).

- Click on a help box (to obtain help information) ;
- Click on a undo box (to undo the last transaction performed on the considered window) ;
- Click on a close box (to close a window and perform associated treatments if necessary).

◊ Actions on an icon :

Managed by THESEUS autonomously :

- Click on an icon and while pressing continuously on the mouse button, move the mouse (to move the icon).

Managed by particular functions :

- Click once or twice on it (to activate the functionality attached to this icon) ;

This is valid for the version of THESEUS running on the GEM window manager. Indeed, if the considered window manager is the "Presentation Manager" of IBM the role of the icon is to embody in-course processes. In this case, the possible actions on the icon are : move the icon, select the icon and open the window corresponding to this icon by a double click on it.

◊ Actions on a button :

Managed by THESEUS autonomously :

None.

Managed by particular functions :

- Click on a button (to activate the functionality attached to it).

◊ Actions on a menu title :

Managed by THESEUS autonomously :

- Click on a menu title (to display the corresponding menu item).

Managed by particular functions :

None.

◊ Action on a menu item :

Managed by THESEUS autonomously :

- Click on a set of displayed menu items and while pressing continuously on the mouse button, move the mouse (to make the menu and to make the menu contents constant somewhere on the screen).

Managed by particular function :

- Click on a menu item (to activate the functionality attached to it).

◊ Actions on a graphic object :

Managed by THESEUS autonomously :

- Click on a graphic object by pressing continuously on the mouse button and move the mouse to a desired area (to display the object in the desired area).

Managed by particular functions :

- Click once or twice on a graphic object (to activate the functionality attached to it).

◊ Actions on an input text object :

Managed by THESEUS autonomously :

- Click somewhere on the text object (to position the cursor) ;
- Input a value in the text object. After this, the value is seized by THESEUS which performs a syntactic control (by reference to a pattern). In the future, it should also be possible to perform such a control character by character. If there is a detected problem, THESEUS displays an error message autonomously;
- Move the cursor in the text object by using key arrows (to move the cursor and to scroll the visible area when the cursor reaches one of the edges of the field and informations are hidden in that directions).

Managed by particular functions :

- Input a value in the text (to activate the function attached to it) ;

The functionality attached to this action could for example performs more complex syntactic controls that these performed automatically by THESEUS.

◊ Actions on an output text object :

Managed by THESEUS autonomously :

None.

Managed by particular functions :

- Click on it (to activate the functionality attached to it).

As the three other generic interactive objects are currently under examination, it is only possible to evoke actions that will probably be implemented in the future. Consequently, the actions proposed here may be submitted to a fundamental revision.

◊ Action on a dialogue box :

THESEUS should support the moving of a dialogue box but beside this, it should only be possible to action the constitutive objects of this box. Among the envisageable actions there should be the filling of a text object, the clicking on a button and so on.

◊ Actions on an scrollable list :

Managed by THESEUS autonomously :

- Click on a scroll bar (to scroll the visible area).

Managed by particular functions :

- Click on one or more values of the displayed list (to activate the functionality attached to them).

◊ Actions on the cursor:

Managed by THESEUS autonomously :

- Move the cursor by using the mouse ;

In the future revision of THESEUS, it should also be possible to reshape the cursor. But this would be done after a request from one of the functionalities attached to actions performed on an object.

Managed by particular function :

None.

To conclude the enumeration of generic actions, it can also be remarked that when a user clicks outside an instantiation of a generic interactive object, THESEUS treats autonomously this action. In fact, it produces a sound in order to indicate that an illegal action has been performed. The fact that user can click anywhere and receives a corresponding THESEUS answer satisfies the lack of precondition associated to the input of the so-called "input translation function" of the Section 5.2.1.

BIBLIOGRAPHICAL REFERENCES

[Apper 83] M. Apperley and R. Spence
The Role of a User's System Model and its Relevance to User Interface Management
in User Interface Management System, Proceedings of the Workshop on UIMS, Springer Verlag, 1983.

[Bass 85] L.J. Bass
A Generalized User Interface for Applications Programs (II)
in Communication of the ACM, Vol 28, n°6, June 1985.

[Bod 89] F. Bodart et Y. Pigneur
Conception assistée des Systèmes d'information Méthode- Modèles- Outils
Masson, Paris, 1989.

[Brown 88] C. Marlin "Lin" Brown
Human-Computer Interface Desing Guidelines
Ablex Publishing Corporation 1988.

[Card 83] S. Card, T. Moran and A. Newell
The psychology of Human-Computer Interaction
Lawrence Erlbaum, Milsdale 1983.

[Chan 88] M. Chandelon
Implementation du modèle APEX
FNDP, 1988.

[Cout 87] J. Coutaz
The Construction of User Interfaces
First European Software Engineering Conference, Strasbourg, France, September 1987.

[Fau 82] B. Faulle
L'informatique conversationnelle, méthodologie d'analyse et de programmation
Les éditions d'organisation, Paris, 1982.

[Fisch 89] G. Fischer
Human-Computer Interaction Software : Lesson Learned, Challenges Ahead
in IEEE Transactions on Software Engineering, January 1989.

[Guru 87a] *Guide de programmation avec GURU*
ISE-CEGOS, Les Edtions du Logiciel, Mars 1987.

[Guru 87b] *Guide d'apprentissage de GURU*
Version 1.0, ISE-CEGOS, Les Editions du Logiciel, avril 1987.

[Har 86a] P. Harmon and B. Sawyer
What is a mid-sized tool ?
in Expert Systems Strategies, Vol.2,n° 4, April 1986.

[Har 86b] P. Harmon
An overview of Knowledge representation
in Expert System Strategies, Vol 2, n°7, July 1986.

[Har 86c] P. Harmon
Objects and inheritance
in Expert System Strategies, Vol 2, n°7, July 1986.

[Har 86d] P. Harmon
Large hybrid tools : ART, KEE and KNOWLEDGE CRAFT
in Expert System Strategies, Vol 2, n°7, July 1986.

[Hart 89] R. Hartson
User-Interface Management Control and Communication
in IEEE Transactions on Software Engineering, January 1989.

[Hix 89] D. Hix and H.R. Hartson
Human-Computer Interface Development : Concepts and Systems for its Management
in ACM Computing Surveys, Vol 21, n° 1, March 1989.

[Hols 87] C.W. Holsapple and A.B. Whinston
Business expert systems
Irwin, USA, 1987

[Hurl 89] W.D. Hurley and J.L. Sibert
Modeling User Interface-Application Interactions
in IEEE Transactions on Software Engineering, January 1989.

[IBM 87] *Common User Access, Panel Design and User Interaction*
IBM, December 1987.

[K-EXPERT 1] *K-Expert Dokumentation*
ADV/ORGA, Wilhelmshaven, mai 1988.

[Kar 83] G. Karnas et P. Salengros
L'ergonomie : adapter ?
in La Revue Nouvelle, n°3, 1983.

[KEE 1] *KEE 3.1. User's Guide and Technical Manual*
IntelliCorps, Inc, Publication Number K3.1-IG-SUN-1, Vol 1 & 2, March 1983.

[Löw 88] J. Löwgren
History, state and future of user interface management systems
SIGCHI Bulletin, Vol 20, n°1, July 1988.

[M1 87a] *M1 Reference Manual for Software*
Version 2.1., Framentec S.A., 1987.

[M1 87b] *M1 Sample Knowledge System Notebook*
Framentec S.A., 1987.

[Mac 83a] M. Mac an Airchinnigh
Report on the User's Conceptual Model
in User Interface Management System, Proceedings of the Workshop on UIMS, Springer Verlag, 1983.

[Mac 83b] M. Mac an Airchinnigh
A model of a User's conceptual Model of...
in User Interface Management System, Proceedings of the Workshop on UIMS, Springer Verlag, 1983.

[Mou 89] B. Moulin
Ingeniérie des connaissances et informatique cognitive des organisations.
Note de cours, FNDP Institut d'informatique, Namur, 1989.

[Myers 89] B.A. Myers
User-Interface Tools : Introduction and Survey
in IEEE Transactions on Software Engineering, January 1989.

[NEXPERT 87] *NEXPERT Object Fundamentals*
Neuron Data Inc, Version 1.1, USA, 1987.

[Norm 86] D.A. Norman and S.W. Draper
User Centered System Design Nex Perspetives on Human-Computer Interaction
Lawrence Erlbaum Associates Publishers, London 1986.

[Olsen 83] D.R. Olsen
Presentational, Syntactic and Semantic Components of Interactive Dialogue Specifications
in User Interface Management System, Proceedings of the Workshop on UIMS, Springer Verlag, 1983.

[Pars 88] K. Parsage and M. Chignell
Expert systems for experts
John Wiley and Sons Inc, 1988

[Rasm 86] J. Rasmussen
Information processing and human-machine interaction- an approach to cognitive engineering
Elsevier 1986.

[Sac 88] B. Sacré
Programmation sous Windows
FNDP, avril 1988.

[Sac 89a] B. Sacré
Eléments d'architecture d'une application interactive sous DecWindows
FNDP, février 1989.

[Sac 89b] B. Sacré
Eléments de spécification d'une application interactive
FNDP, 1989.

[Sca 87] D.L. Scapin
Guide ergonomique de conception des interfaces homme-ordinateur
Rapport INRIA, France, octobre 1988.

[Shnei 87] B. Shneiderman
Designing the User Interface : Strategies for Effective Human Computer Interaction
Addison Wesley Publishing Company, 1987.

[Stru 83] H.J. Strubbe
Components of Interactive Applications
in User Interface Management System, Proceedings of the Workshop on UIMS, Springer Verlag, 1983.

[THESEUS 1] D. Eckardt, W. Hübner, G. Lux-Mülders and M. Muth
THESEUS, the software Engineering User Interface
ZGDV, Darmstadt, Federal Republic of Germany.

[THESEUS 2] W. Hübner, G. Lux-Mülders and M. Muth
Designing a System to provide Graphical User interfaces : the THESEUS approach
ZGDV, Darmstadt, Federal Republic of Germany.

[THESEUS 3] M. Muth and T. Neumann
Das UIMS THESEUS
ZGDV, Darmstadt, Federal Republic of Germany ;
ADV/ORGA, Wiesbaden, Federal Republic of Germany.

[THESEUS 4] W. Hübner, G. Lux-Mülders and M. Muth
THESEUS , die Benutzungsoberfläche der UNIBASE-Softwar entwicklungsumgebung
Springer Verlag, Berlin, 1987.

[THESEUS 5] H. Dümcke
Konzepte Anbindurng an die Anwendung Protokoll
ADV/ORGA, Wiesbaden, Federal Republic of Germany, May 1989.

[THESEUS 6] *THESEUS-Pmr Review der Anfurderungen an THESEUS-Ergebnisprotokoll*
ADV/ORGA, Wiesbaden, Federal Republic of Germany, July 1989.

[THESEUS 7] R. Prem
THESEUS Konzepte Objektverwaltung in THESEUS
ADV/ORGA, Wiesbaden, Federal Republic of Germany, May 1989.

[THESEUS 8] C.P. Brück and M. Wigger
Werkzeuge für Dialog-und Formularentwurf mit THESEUS als User Interface Management System
ADV/ORGA, Wiesbaden, Federal Republic of Germany, (slides).

[THESEUS 9] D. Eckardt, W. Hübner, G. Lux-Mülders and M. Muth
THESEUS : The Software Engineering User Interface Management System
ZGDV, Darmstadt, Federal Republic of Germany.

[Van 87] A. van Lamsweerde
Méthodologie de développement de logiciel
Note de cours, FNDP Institut d'informatique, Namur, 1987.

[War 88a] G. Warnant
Methodologie de développement d'une application interactive
Rapport de synthèse, FNDP, septembre 1988.

[War 88b] G. Warnant
Methodologie de développement du dialogue d'une application interactive
FNDP, 1988.

[War 88c] G. Warnant
Elements de modélisation du dialogue d'une application interactive
Note de cours, FNDP Institut d'informatique, Namur, mai 1988.

[Woo] D.D. Woods
Cognitive Technologies : the Design of Joint Human-Machine Cognitive Systems
in the A.I. Magazine.

[Youn 88] M. Young, R.N. Taylor and D.B. Troup
Software Environment Architectures and User Interface Facilities
in IEEE Transactions on Software Engineering, Vol. 14, n° 6, June 1988.