INTELLIGENT GRAPHIC INTERFACE
CAPTURING RULES OF HUMAN-COMPUTER INTERACTION IN A KNOWLEDGE BASE

by

PEDRO MANUEL BARBOSA FERRAZ DE ABREU

Licenciado, Computer Engineering
New University of Lisbon (1983)
Bacharel, Electronics and Telecommunications
Superior Institute of Engineering of Lisbon (1981)

Submitted to the Media Arts and Sciences Section in Partial Fulfillment of
the Requirements of the Degree of Master of Science

at the

Massachusetts Institute of Technology

September 1989

© Massachusetts Institute of Technology 1989. All rights reserved

Signature of Author

Certified by

Muriel Cooper
Professor of Visual Studies
Thesis Supervisor

Accepted by

Stephen A Benton
Chairman
Departmental Committee on Graduate Students
The objective of the thesis is to present a new approach to build graphical user interface systems, within the domain of research for intelligent interfaces.

The problem under consideration is the lack of flexibility of current interface systems to adapt "on-the-fly" to a new context (automatically, or by user specification), in a dynamic environment. A new context occurs when different type of data, a different kind of user or a different goal surfaces, and usually implies re-writing the interface code.

The new approach consists of identifying consistent "chunks" of interface knowledge (set of facts, rules and procedures characterising human-computer interaction), finding the adequate representation for each and integrating them in a knowledge-base system structure, together with a graphical interpreter.

The goal of this approach is a highly adaptable interface, with learning potential.

The presentation of this approach is made through the description of a model, and through the implementation of a small prototype.

The prototype handles simple examples of how it is possible to represent human-computer interactions in a knowledge base; how the interface can adapt to a different context; how changing a rule changes the interface; and how it is possible to change the interface at run-time.

The examples were taken from a scenario with a varying context of two different users and two different simple applications - a small mapping system, and an experimental medical expert system.
INDEX

1. The Problem 4

2. User interfaces: problems and evolution 9
   2.1. Application-dependent interfaces
   2.2. Interface management systems
   2.3. Towards intelligent interfaces
   2.4. Where it makes a difference
   2.5. Current scenario

3. The approach 18
   3.1. The "Columbus egg"
   3.2. The Interface knowledge
   3.3. The Knowledge-based structure
   3.4. A Knowledge-based graphical interface system
   3.5. The prototype
   3.6. The interface knowledge in the prototype
   3.7. The problems
   3.8. Current work with a similar approach
   3.9. Next step, Learning

4. The experiment 34
   4.1. The objectives
   4.2. Examples: a mapping system
   4.3. Examples: a medical expert system
   4.4. Examples: Conclusion

5. The tools 61
   5.1. The first options
   5.2. The inference engine

6. Code components and resources used 67

Acknowledgements 68
Glossary 69
References 71
1. The problem

The focus in this thesis is on the following class of problems: what happens when we need - or want - to change the interface, to adapt it to a new context? How can we minimize the effort needed to meet this requirement while maximizing efficiency and user satisfaction?

We can define, informally, the context of a user interface, as something characterized by the user, the application, and the set of resources involved in a human-computer interaction at a given moment; or, more formally, as "a 2-tuple {user-type x, application-type y}" [Ferraz de Abreu 87], where the application-type is a n-tuple { S_1 {application i}, S_2 {system-resource i} }, with n=k+m. (In fact, the more formal definition will help to simplify the proposed interface model).

In other words, a new context can occur when different type of data, a different type of user or a different goal surfaces. Let us consider a few examples of plausible contexts:

**Context 1** - Suppose we are civil engineers developing an application to study possible paths for road construction in the White Mountains. The data available is the E.T.M. (Elevation Terrain Model) of the area. We want an interface that will allow a graphic query to this data, using a stylus to select from a menu of commands. Someone familiar with the graphic window system available (we are not all computer hackers ...) creates an interface with windows to display bitmaps (for E.T.M. data), and word-windows that constitute the menu and act as buttons to activate commands.
Variation 1 - Everyone is happy, till the moment that new data arrived: bitmaps with feature data for the same area (water, roads), and a map with data on surface temperature. Now we need to update the graphic interface with new windows to display the new bitmaps, that have to be made transparent to allow overlay, and with new commands to display or hide the new data on the screen. This usually implies to re-write the application code to handle the new cases, even if they are relatively simple. And what if the person familiar with the window system left? We have no choice but to get into the lower level code and study it till we know how to program it.

Variation 2 - Suppose that it is decided that for a quick, qualitative analysis it is preferable a graphic representation for the values of Elevation and Surface Temperature (like bars or thermometers), instead of the simple number previously displayed. This will imply to recode the output functions associated with each type of data, or duplicate (or triplicate) them, for each type of representation desired.

Variation 3 - Suppose now that we managed to add the last of the new commands and windows, just to find out that now, when all data available is displayed, there is not enough room on the screen for data and commands to be visible simultaneously - a rather inconvenient thing to happen. We will have to go back to the application code and reorganize the screen layout, or the presentation of each element to be displayed, or both.

The issue here is to have the decisions on how to represent and display each type of data (values, legends, comments, pictures, etc) made by the interface system, without the need to have it re-coded for each application program, or for each set and type of map data.
Variation 4 - When things seemed to stabilize, a new team joined the department, and wanted to use the same data, the same application, but with a different goal: to study possible paths for radio-transmission links. While for road construction the relevant information was a) continuous elevation data, b) land use, c) surface material, for radio-transmission links the relevant data will be a) Z values for terminal locations, and b) all points in the line-of-sight that have greater Z value.

The issue here is how to filter the type and presentation of data to display according to the particular user view, or goal, to avoid either "data overdose" problems, either unnecessary duplication of menu commands - which again will imply to re-code the interface to include this new goal.

Context 2 - Let us consider now the case of an expert system for medical diagnosis. Several systems of the kind were developed in this domain (the most famous was MYCIN), and continue to be the object of intense research, particularly at MIT A.I. Lab. Therefore, they provide a relatively solid ground to describe a plausible scenario. An interface for such a system, could be something like this:

![Figure 1](image_url)

A possible interface for our bright medical expert system
On the screen, we have an image of a human body (previously selected according to user gender), where the user points the location of his problems. The user pointed to the nose and the system displayed several "cross-filling" yes-or-no boxes, to describe symptoms: "running nose?" "sneezing?" "itching?". The user filled in the respective boxes, and the system diagnosed a strong allergy, with no fever expected. (You could say that for this you don't need a medical expert system, but more complicated diagnosis procedures are similar).

It is also quite straight-forward to realize that, supporting this process, we have an expert system containing a set of rules representing medical knowledge, and some graphic code incorporated to display the boxes, etc.

Variation 1 - Suppose now that you are a critical M.D. expert and say: "AhAh, the system should also ask for 'coughing?', because it could be a strong flu, with fever!". What would have to be done? First, you have to change (or add) the appropriate medical rules. That is simple, since in a rule-based system, you can change rules without needing to change code (inference engine). But second, you have to change the code (so that it will display the extra box, and read the user answer) and recompile it.

![Medical expert system improved with "coughing?" symptom query](image)

Fig 2

Medical expert system improved with "coughing?" symptom query
Variation 2 - But suppose again that, later on, a M.D. specializing in tropical diseases comes and says: "AhAh, dear colleague, just answering yes-or-no on 'fever?' is not enough; if fever is between 37.5 and 38 degrees centigrade, the patient does not need any special therapy; but between 38 and 42, that's another story!"

And now, not only you have to modify/add medical rules, but also change the code to display, for example, a "dialogue" thermometer, instead of a yes-no box. Then the code would have to be recompiled.

Variation n - And what if another user finds a thermometer using a centigrade scale too confusing? Or if another one prefers to type a number? Or if a frequent user finds tiring the text legends on the boxes, and prefers icons of "running nose" "coughing", etc? Or if another symptom is found? Or if the number of new boxes added implies reorganizing the whole screen layout, resizing the boxes? There is simply no way to guess in advance all possible important modifications or useful alternatives to be offered.

In general, we can conclude that this system has a component which is easy to modify - the medical knowledge - and another one which is more painful to change or extend - the graphical presentation of the input/output.

Finally, another issue is how to extensively use, within a particular application, elements of the interface system developed for another one. We shouldn't have to start from scratch an interface, everytime a new application is born.

These are the issues identifying the class of problems under consideration.
2. User interfaces: problems and evolution

There are multiple possible ways to classify the work done in this field, and probably as many to identify its major historical phases; each different way reflects the different nature of the problems under focus. Two good examples are Baecker's "state-of-the-art review" [Baecker 80], mainly hardware oriented, and Myer's taxonomy [Myers 86] with focus on program styles and features.

From the perspective of this class of problems, it is helpful to visualize the evolution of user interfaces (even at the risk of some inevitable over-simplification), roughly according to 3 generations:

2.1. Application-dependent interfaces

In the first generation, interface and application are usually together, as parts of the same program. It follows that the solution to our problem implies changing the program's code, that is, to restart the cycle "code-edit-compile-debug", each time we introduce a change in the interface.

This is obviously not a satisfactory solution. Any user interface is a kind of a language, through which the user and the program (or programs) dialogue. And after all, with a "common" computer language (lisp, pascal), no one would find it reasonable to have to change the language compiler/interpreter, each time a program is modified!

With the advent of new programming techniques, such as "modular programming" and the use of data-structure "encapsulation" (abstract data types) [Dijkstra, Parnas], it became possible to minimize these problems in a
certain extension, using separate modules for interface functions and constraining the interaction with other parts of the code. But these were weak improvements, while made under the scope of an "application-dependent" interface model.

Fig. 3
An "application-dependent" interface model

2.2. Interface management systems

This led to a second generation, where the emphasis shifted towards the development of tools to generate and evaluate interface systems. Among several designations, such as "dialogue management systems" [Roach 82], or "abstract interaction handlers" [Feldman 82], the one that became more commonly accepted was "User Interface Management Systems" (UIMS) [Kasic 82].

Fig. 4
An independent interface model:
Interface generated by an User Interface Management System
A typical UIMS (Fig. 5) is composed of two parts: a pre-processor (interface generator) and a run-time support environment (interface manager and evaluator). The pre-processor can contain: a module library (each module is a piece of interaction features, or a "dialogue cell"), a module builder, and a glue system (to bind the selected modules to each other and to the application). This way, different interfaces can be generated by the same system, using different combinations of these prepackaged modules.

Although this represents an enormous advance, there is a major limitation: the system does not support modifications at run-time. This means that, even for a minor modification to the interface, someone familiar with the system and the computer environment will have to restart the cycle "module building - module selection - glued system compilation", or even to change or extend some part of the code. The system is still not easily adjusted by the end-user, and certainly not able to adjust itself.

2.3. Towards intelligent interfaces

This is the reason why I think it makes sense to speak about a third generation
of user interfaces. Given the nature of the limitations of UIMS, serious improvement (again, strictly from the point of view of our problem) is only possible by adding a new dimension: \textit{interface intelligence}.

The foundations for this new generation can be seen already in several past and recent works, in domains such as: \textit{direct manipulation, program visualization, visual (or graphical) programming, presentation systems, event response systems, programming by rehearsal, programming by example, context understanding, and user models}. Without getting into details [definitions included in glossary], it is important to mention the following:

- The work done in the first four areas (\textit{direct manipulation, program visualization, graphical programming, presentation systems}), although not directly related to introducing the intelligence factor, is responsible for significant improvements in the \textit{graphical mode} of interaction. Why is this so important for intelligent interfacing? Two main reasons: with the graphic mode, we have a greater potential of communication, and it is easier to communicate at a higher level. Zdybel and Myers, among many others, refer to this:

"Graphic output is the best way to communicate a substantial amount of information to a human user because it exploits the high-bandwidth human visual channel. Graphic input (...) is an extremely economical way to describe something" [Zdybel 82]

"Another motivation for using graphics is that it tends to be a higher-level description of the desired actions (often de-emphasizing issues of syntax and providing a higher level of abstraction)" [Myers 86]

Poltrock, Steiner and Tarlton worked on the development of graphic interfaces for knowledge-based systems [Poltrock 86]. This work helped to emphasize the power of graphics to deal with \textit{knowledge representation}. But an important aspect is that the knowledge-based systems are still viewed by them as a \textit{target application} for the graphic interface, and not as the \textit{core} of the interface system itself.
The fast expansion of object-oriented programming [Stefik 84] among the user interface research community, is another important factor. Since object-oriented languages are already a kind of knowledge-representation language, they form a bridge between "traditional" programs and knowledge-based systems. Systems like EZWIN are introduced as "an editor for presentation objects ... (where) we can supply a library of predefined command and presentation types" [Lieberman 85], and had a significant influence in the development of the ideas presented in this thesis.

- The other referred areas (event response systems, programming by rehearsal, programming by example, context understanding, user models) bring us to the edge of intelligent interfaces.

In event-response systems (ERS), the mechanisms of interaction are represented in IF-THEN rules: events set/reset boolean flags, triggering the execution of operations and/or the set/resetting of other flags. Ralph Hill implemented one of these ERS [Hill 87]. His system is a demonstration that, at least in some degree, it is possible to capture some mechanisms of interaction under a rule format. But the system is not adjustable; his goal was to build a fast system to support concurrent actions, and he opted for a built-in compact interpreter, instead of an inference engine, for the control structure. Therefore, a change in the interface implies the redesign of the system.

With programming by rehearsal, or by example, we obtain a kind of learning mechanism. This is particularly true for systems that try to infer a general program structure from several examples describing an algorithm. Bauer [Bauer 78] presented a simple system that attempted this through generalization (the program decides which values should be constants and which should be variables). The learning ability is naturally crucial for Intelligent Interfaces; unfortunately, these systems are still very limited in scope and not very reliable outside their experimental micro-world. Other
interesting systems are Tinker [Lieberman 82], Graphical Thinglab [Borning 86] and Peridot [Myer 87].

Context understanding systems are another approach to keep in mind. I include in this category not only systems that have a natural language understanding ability (since natural languages are languages generated by context-sensitive grammars [Monteiro 81]), but especially multi-media systems (or multi-channel), such as PUT-THAT-THERE [Bolt 84]. An important message to retain from this system is that a) an intelligent interface should convey the feeling that the computer is making a serious effort to understand the user goals, and b) a multi-channel communication is likely to facilitate even higher level dialogues.

But how can we improve the system's ability to understand the user goals? A favorite approach consists of building user models. We can refer to Rich's user"stereotypes" to predict behaviour (goals, general motivation, common confusions and expectations), leading to the notion of "canonical user" [Rich 83], or to Quinn's approach:

"There are two aspects of a strong user model: (1) the system must have knowledge about characteristics of the typical user as well as about those of the current individual user; (2) the system must contain a 'behavioral semantics' -- a means by which it can make the appropriate associations between its knowledge about users and the in situ behaviour of the user" [Quinn 86].

Card, Moran, and Newell [Card 80] introduced an interesting model to account for the text editing behavior of experts performing routine tasks: the GOMS model. In this model, the expert knowledge representation consists of four components: Goals, Operators, Methods, and Selection Rules. Goals are what the expert user is trying to accomplish; operators are the actions the system allows to accomplish the goals; methods are the sequences of actions that will accomplish a goal; and selection rules are the conditions under which a given method is chosen to accomplish a given goal. Kay and Black [Kay 85] also
worked to develop this model.

Philip Barnard, with his "Cognitive Task model (CTM)" [Barnard 87], prefers to focus on characterizing user behaviour, as a result of a certain state (configuration):

"Under conditions
<Config X>
<Key attributes of procedural knowledge>
<Key attributes of record contents>
<Key attributes of dynamic control>
Result:
<Key attributes of user behaviour>"

["Cognitive Task model (CTM)""]

An important aspect of these models (particularly GOMS and CTM), is that they point to the conclusion that (at least in a certain extent) it is possible to characterize user behaviour with a rule representation format.

Odd as that might seem, none of these works dedicated attention to what appears to be the next logical step: application of these user models to a Knowledge-Based Interface System. That might be in part explainable by the fact that most of these authors concentrate on the aspects related to cognitive science, and therefore are more interested in developing simulation environments to test cognitive models of the user.

Finally, there isn't any work, to the best of my knowledge, on the other end of an interface - the application' space; no "application models" where developed to coexist with these user models. I claim, however, that they are also important and useful; and that in order to develop them, we need to build a comprehensive Taxonomy of applications, based on clearly specified attributes.
2.4. Where it makes a difference

It can be easily seen that these 3 generations overlap in time; the division is more conceptual (families of interfaces) than chronological. Also, it is fair to mention that in many cases (within a limited range of expected variation), a UIMS can probably generate an efficient interface; and sometimes (e.g. when speed is THE factor), building an application-dependent dedicated interpreter is even better. But overall, there is a much wider range of applications where an Intelligent Interface will make a difference.

As an example, it is enough to think that practically all applications can be viewed in many different ways by separate users, or by the same user at different times - therefore, with many different interface requirements. For instance, Olsen, Buxton, Kasic and others [Olsen 84] identified "(...) nine distinct roles in the development of an interactive application. These roles are: End user, application analyst, application programmer, dialogue author, graphic designer, dialogue evaluator, UIMS builder, environmental designer, and environment evaluator."

More generally, this approach will make a difference each time we face a very dynamic context.

2.5. Current scenario

Naturally this overview does not cover all important research in the field. The issues referred to were the main ingredients that contributed to this approach (importance of the graphic component, the learning ability, the rule representation of interaction, the multi-channel potential, the rule representation in user modeling). What about the current scenario on specific work in Intelligent Systems?

Baecker and Buxton, in their 87' edition of "Readings in Human-Computer
Interaction", resume the overview on Intelligent Interfaces this way: "The attempt to embed intelligence in interfaces is in its infancy, and is most often applied in one of two specific domains". The two specific domains referred to are "intelligent help systems" [Rissland 84] and "intelligent tutoring systems" [Anderson 86].

Significantly, this overview on Intelligent Interfaces is included in the chapter "Research frontiers and unsolved problems" of the quoted "Readings...". In fact, most of the efforts in this domain are still on user modeling.

This is indeed the land of unsolved problems.
3. The approach

3.1. The "Columbus egg" ("Ov o de Colombo")

It is said that, after Colombo's discovery of the American continent under the Spanish colors, many nobles serving the King of Portugal tried to minimize the event: "Big deal! Anyone could come up with the idea of navigating west until they found something", they whispered in the noble corridors. They were not indifferent to the fact that Colombo, although born in Genova, was living in Portugal and had tried without success to convince the King to sponsor him. Thus, he offered his services to the Queen of Spain, bringing her glory and power... Then one day, Colombo met with the nobles and, showing them an egg, asked if they had any idea of how to keep it still, vertically, on a table. They tried and tried, but the egg always would roll to one side; they said: "it is impossible! ". Then Colombo hold the egg, made a small dent in the base, and the egg did keep still. The nobles said "haha! but that way, it is obvious, anyone could have made it"; to what Colombo replied, "yes, but I was the one that did it". And the tale of the "ovo de Colombo" was kept in the memories of the kingdom for ever after.

One and a half year ago, when I started putting together the ideas that eventually became this thesis, the most perturbing question was: "why isn't anybody else using this approach, it seems so obvious!" It seemed indeed that it was a kind of "Columbus egg" idea.

Why, all the key ingredients were there. From old Formal Languages' finite state automata theory (basic formalism supporting event-response systems), to more recent Cognitive Science's user model paradigm, there was growing evidence in favor of the feasibility of a whole rule-based human-computer interaction model. From Artificial Intelligence's expert-systems and learning sub-domains, we were obtaining an increasingly clear view of the power and flexibility of a knowledge-based system structure to handle rule-based models. Putting all this together seemed more simple than making a dent in an egg!

Half an year later, I found out that a) the idea was not so simple, there were
several problems (and limitations) associated with it; b) There were indeed some people exploring a similar track ([Foley 88], [Tyler 88]), although with significant differences. This was first a disappointment (I was not the only "Colombo"), then a reassuring element (having experimented researchers like Foley exploring on similar ground), and finally a source of inspiration to give a more definite shape to my approach. First, I'll introduce the approach; then, I'll refer to the problems and the new work.

3.2. The interface knowledge

The first component of the approach is, then, based on the following idea: Since some components of human-computer interaction can be represented in a IF-THEN rule format (e.g. user behaviour), then let's try to capture all the key components, represent them in the most adequate mode (not necessarily in rules), and integrate them in a common system.

As a matter of fact, one of the key supports for this idea comes precisely from the UIMS generation.

A very important premise is behind the UIMS structure: that the mechanisms of interaction, of dialogue, between the user and an application can be separated from the application functionality, i.e., those mechanisms can exist independently. In other words, UIMS are based on the premise that "the lexical and syntactic components can be isolated from the application semantics" [Tanner 83].

As a first step in defining my thesis, I extend the scope of this idea to define the following sub-thesis:

sub-thesis 1: It is possible to capture a consistent set of rules and procedures characterizing the human-computer interaction, from both user and
application spaces (considering a finite set of pairs \{user-type x, application-type y\}).

I call this set of rules and procedures \textit{Interface knowledge}.

According to this sub-thesis, we can model a "user interface as an independent system that incorporates knowledge about both the user and the application, through its virtual intersection" [Ferraz de Abreu 87]

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{interface_model.png}
\caption{An interface as the virtual intersection of the user and application spaces}
\end{figure}

Rissland, for instance, agrees that "For an interface to be intelligent (...) it must have access to many different sources of knowledge" [Rissland 84]. She identifies 7 of these sources: knowledge of a) user, b) user's tasks, c) tools, d) domain, e) modalities of interaction, f) how to interact, and g) evaluation.

According to my model, I introduce a slightly more compact and simplified view, identifying 3 components in the "interface knowledge": knowledge about the \textit{user}, knowledge about the \textit{application} and knowledge about the \textit{interaction mechanism}.

The knowledge about the \textit{user} can refer to a) user level of expertise; b) user area of expertise; c) user personal preferences and style.
The knowledge about the *application* can refer to: a) type or class of application (e.g., simulation, information retrieval, calculus, graphic design, etc); b) type or level of interactivity (input/output); c) type of tasks (e.g. add, multiply, or rotate, scale, etc); d) associated tools (e.g. color palette for graphic design).

The knowledge about the *interaction mechanisms* can refer to: a) control of user input (e.g. high level stylus handler); b) control of output (e.g. generation of different presentations, graphic rules [Mackinlay 86], or screen layout); c) control of interaction between interface entities (e.g. messages between objects); d) control of internal states (e.g. navigation in a menu-tree).

### 3.3. The knowledge-based structure

The second component of the approach is based on the following idea: Since we can identify and represent something like the interface knowledge, then let's use a Knowledge-based system (KBS) structure as the *core* itself of the interface system. The immediately obvious advantage is that, by the proper nature of a KBS, we can change its contents (the knowledge) without changing any program code (inference engine); therefore, we should be able to change the interface easily, and also at run-time. A second advantage, although not so immediate, is that a KBS is the proper structure to support learning mechanisms; therefore, we should be able to develop a learning ability in our interface system.

Also, we know that it is possible to develop some kind of graphic interface for a knowledge-based system. We should be able, then, to articulate our Knowledge-based interface system with a graphic interpreter.

Accordingly, as the second step in defining my thesis, I state the following
Sub-thesis:

Sub-thesis 2: It is possible to use a knowledge-based system structure as the core of a user interface; more specifically, it is possible to integrate the interface knowledge in one KBS, articulated with a graphical interpreter.

3.4. A knowledge-based graphical interface system

The motivation for the new approach is to produce an interface system that, within a stable structure and without the need to re-write its code, is able to:

- support user-defined modifications, at run time;
- adapt to a variety of contexts (different users, different views of an application, different applications);
- infer a correct context-dependent sequence of operations from stated user goals;
- have a learning potential.

According to the previous steps, I state my thesis the following way:

Thesis: Mapping Interface knowledge to a knowledge base structure is a good framework for an highly adaptive, potentially intelligent interface.

3.5. The prototype

A small prototype was implemented, using a more detailed model. The implementation of the prototype was based on my previous work, in particular the first steps of the development of the SmartMenu system. The current
presentation is also based on a previous paper describing that work [Ferraz de Abreu 88].

The prototype system consists of a small menu-driven knowledge-based interface generator. Iconic menu-type interfaces can be created or modified using a restricted set of meta-commands (special menu) and a pool of system resources (elementary functions, graphic elements and data structures), together with the application's own operations and data structures.

In fact, making use of the abstract data type formalism, and its close relative, the object-oriented concept, all components of the system are considered Menu-Objects. These are divided into 4 super-classes: Meta-command-object, Command-object, Application-object and Screen-object. More specifically, each object integrates a data structure with its associated operations (procedures, predicates, guarded commands, daemons), and is part of a network providing inheritance mechanisms (procedural and data inheritance).

The meta-commands include: object manipulation (create-interface-object, write-object-parameters, change color), rules manipulation (show, create, delete, change and clone rules), knowledge-base control (trace inferencing, start inferencing, show current status) and application management (load application, reset, quit, clear, break).

The pool of system resources include: color-palette, icon-library, and some presentation-objects (string, number, bar, thermometer, transparent bitmap, etc), with related functions (e.g. change-object-color, change-internal-layout, etc).

The command-objects are user-defined, and constitute the menu interface body.
The application-objects represent the application body, viewed from the interface.

The screen-objects are compound hybrid objects, i.e. sets of meta-command, command and application objects that are displayed simultaneously.

Conceptually, the system can be described as an organized collection of Menu-Objects, a set of Rules shaping user-object and object-object interaction, an Inference Engine and a Working Memory with the current context and history.

Most Menu-Objects have a dual nature. They are knowledge-base elements from the system view, but they are also presentation objects from the user view [Zdybel 82] [Cicarelli 84] [Szekely 87]. The screen handling and graphic functions necessary for the object presentation are provided by a Graphic Interpreter, in a transparent manner for the user and the K-B System (it is only "seen" by the objects).

Model for a knowledge-based graphical interface system
3.6. The interface knowledge in the prototype

To define a kind of user model, we need to describe: the level of expertise dealing with the interface (novice, average, expert); the style of user preferences (icons with text, "true" icons - images - , or both); the user name; the user history (no. of 'undo' or 'help' operations done, etc); and a set of related rules, constituting a small inference net. For example:

rules (user model):
(ex:)

IF user.class {novice}
THEN set legends {on}
  set text-icon {on}

IF user.class {normal}
THEN set legends {on}

IF user.class {expert}
THEN set legends {off}

IF user.style {image-icon}
THEN set image-icon {on}

IF #"undo" operations > 5
THEN set user.class {novice}

IF login {pedro}
THEN set user.class {expert}
  set user.style {image-icon}

(etc)

end of rules (user model)

To define a kind of an application model, we need to describe the application name, class (information retrieval, graphic design); and also a small set of related rules. For example:
rules (application model)

(ex:)

IF appl.name (nynex-advertising) THEN set appl.domain {graphic-design}

IF appl.name (map-darpa) THEN set appl.domain {map}

IF machine.name (Fritz) THEN set machine.type {renaissance}

IF appl.domain {graphic-design} OR appl.domain {graphic} THEN set presentation-type {graphic}

IF appl.domain {map} machine.type {renaissance} THEN set presentation-type {word}

IF presentation-type {graphic} object-type {text-window} THEN set windows {transparent} set highlight {invert-color}

IF presentation-type {word} object-type {icon} THEN set highlight {perimeter}

IF presentation-type {graphic} OR presentation-type {control-panel} THEN set menu-structure {horizontal}

IF presentation-type {word} THEN set menu-structure {tree}

(etc)
end of rules (application model)

26
Defining an equivalent model for the knowledge describing the interaction mechanisms is a bit more complex - and challenging. We can have a working system with limited user and application models; but we need a "real-size" interface knowledge for the interface infra-structure.

First, I considered a possible set of rule templates:

(Object-handler rules)

\[
\text{IF part} \rightarrow \text{State of the system} \\
\text{object-enable (object)} \\
\text{state-button (state)} \\
\{ \text{history (previous state)} \} \\
\text{THEN part} \rightarrow \text{Actions/range} \\
\text{operation(s) on object (object)} \\
\text{set state-menu (newstate)} \\
\{ \text{set history (state)} \}
\]

(menu handler rules)

\[
\text{IF part} \rightarrow \text{State of the system} \\
\text{menu-option (option)} \\
\text{history (previous state in menu-tree)} \\
\text{THEN part} \rightarrow \text{Actions/range} \\
\text{modify/continue level in menu-tree} \\
\text{modify/continue screen display} \\
\text{store history (level, option)} \\
\text{execute operation(s) on object(s)}
\]

(user-goal-definer rules)

\[
\text{IF part} \rightarrow \text{State of the system} \\
\text{history (level, option)} \\
\{ \text{history (level, option)} \}^* \\
\text{THEN part} \rightarrow \text{Actions/range} \\
\text{set user-goal (goal)}
\]
(user-goal-handler rules)

    IF part -> State of the system
          user-goal (goal)

    THEN part -> Actions/range
           store history (level, goal)
           select plan {state, goal}
           execute operation(s) on object(s)

[rule templates]

Then, I developed a set of rules based on these templates, as shown in next chapter.

Summarizing, we can say that, in this system, the "interface knowledge" associated with interaction mechanisms consists of:

a) The information contained by the collection of Menu-objects (meta-commands and commands), as defined above;

b) The information expressed by a set of rules with the described format (object-handlers, menu-handlers, user-goal-definers, user-goal-handlers).

Concrete examples of these types of rules are given with the examples (Chapter 4).

3.7. The problems.

While working on this approach, I have identified a few problems associated with it:

a) There is no simple way to identify and represent the knowledge needed to control a sequence of commands. A certain amount of trial-and-error
experimentation was necessary.

b) Without defining a taxonomy for applications, it is not possible to develop any consistent application models. Therefore, we need a taxonomy, even if a modest, small one.

c) The most adequate knowledge representation for each of the interface knowledge components is not necessarily a rule format. This raises the problem of how to articulate different representations in one single inference net. The problem exists, because to different representations correspond different control systems; for instance: frames use an agenda; scripts, a "blackboard"; rules, use procedural attachment; and first order predicate calculus, means-end analysis. I have concluded that a hybrid system with rules and frames can solve the problem, but this would need further analysis and testing before a final conclusion.

d) We have to keep the interface knowledge consistent. This is no minor problem, since several users will be allowed to modify, add or delete components of the knowledge-base; after all, that is the essence of the power and flexibility of this structure. The "classical" solution for this problem is to filter the knowledge acquisition through a "Truth Maintenance System". My option was to control and constrain the interface knowledge manipulation, through specific knowledge-base management functions.

e) The adequacy of an inference engine to deal with low level interaction operations is questionable. This was the most important problem found. Usually, K-B Systems are prepared to deal with cycles (question-answer) that can not be interrupted and reverted or re-defined. But this is an important requirement for an interactive graphical interface. Also, they are not designed to deal with an event-response system like a stylus-driven (or mouse-driven)
interface system. This leads to the following problems: *speed* (an inference engine is likely to be much slower than dedicated interpreters); and the need for encapsulation of *low-level operations* (such as stylus $x, y$ coordinates location).

This last problem is particularly hard to solve. It is enough to imagine a scenario with an high branching factor of a menu-tree (number of options per level per option). In this case, the system faces an explosive multiplication of low-level rules. This seems to indicate that a "pure" knowledge-based environment (i.e., without using a graphic interpreter for these low-level operations) is not practicable.

For a while I played with the attractive idea of implementing the graphic interpreter also as a package of low-level to high-level rules. These rules would be included in the same KBS and therefore controlled by the same program - the inference engine. This would be attractive not only because of the structure uniformization, but also because it would allow any device-handler module to be included in the same KBS, thereby opening the door for an easy multi-media, multi-channel interface representation.

When I found (confirming other opinions) that this would be a source of many headaches, not to mention a possible dead-end, I adopted the cautious way, if not the most pure: an independent graphic interpreter is in charge of all low-level graphic functions, and most high-level ones.

### 3.8. Current work with a similar approach

Although none of these problems are unsolvable, they constitute good reason
enough to explain why there were not many people following this path. Even so, in the CHI (‘88) two papers shared the concept of the second sub-thesis: the use of a knowledge-based system structure for the user interface. There are important differences, though. Here are some:

- In their work, Foley et al. [Foley 88] didn’t make use of the presentation objects concept; also, changes in the interface are targeted to produce different interfaces but with the same functionality (equivalents) - My approach is intended to allow different functionalities.

- In his work, Tyler [Tyler 88] separates the interface knowledge in several knowledge bases - My approach is to integrate it in an articulate inference net, in only one KB; also, the emphasis is only on the user model.

In both works, there is no emphasis on the graphic component. There is no attempt to deal with high level inference of user goals and sequences of operations. There is no attempt to develop application models, and there is no clear notion equivalent to my sub-thesis 1: the interface knowledge concept.

Nevertheless, they represent a very important reference for this work.

---

**Fig. 8**

Structure of a menu-object, using the presentation-object concept
3.9. **Next step, learning.**

The next logical step, will be to include, in the next version, learning mechanisms in the interface system. Considering that a significant part of the *interface knowledge* has either a taxonomic structure (user model, application model), or a "guarded-commands" structure (event-response rules, interaction mechanisms), my approach is to use the paradigms of "learning by generalization" and "learning by specialization". The reason is that they have heuristics quite appropriate for use with this interface knowledge structure:

In order to *generalize*,
- Change constants to variables,
- Drop conditions ("AND" clauses, from IF part of rules),
- Add options ("OR" clauses),
- Climb in an IS-A hierarchy.

In order to *specialize*,
- Restrict the range of variables,
- Add conditions,
- Drop options,
- Descend in an IS-A hierarchy.

With these heuristics, the system will be able to generate new rules from the given ones. A general algorithm would be: "If no rule matches current context, then search for best heuristic; apply heuristic; if it fails, ask the user". This algorithm can be represented with a set of *meta-rules*.

Experience with knowledge acquisition in rule-based systems (such as MYCIN and others), shows that a good way to avoid complex truth maintenance is to put severe constraints in this process [Davis 77]. One relatively simple
constraint to apply is to define templates that the system has to mandatorily observe: rule models, function templates, and object-schemata.

A good strategy to start with would be:

a) Define rule models and function templates for the interface knowledge;

b) Initially, restrict learning heuristics to: "Drop conditions" and "Add conditions";

c) After testing the system, allow other heuristics, incrementally.
4. The experiment

4.1. Objectives.

The objectives of the experiment were to test and demonstrate:

a) How it is possible to represent human-computer interactions in a knowledge base;

b) How the interface can adapt to a different context;

c) How changing a rule changes the interface;

d) How it is possible to change the interface at run-time.

The modifications may consist of changing the interface presentation (screen layout, menu-object type, shape, size and color), functionality (operations activated by menu-objects) or structure (hierarchy of menu-objects, sequence of commands).

In future work, I intend to test and demonstrate:

e) How strategies (sequence of operations) are inferred from user-stated goals;

f) How the interface can learn (generate new rules), through generalization or specialization of the given set of rules.

In order to provide a testbed for the system, two small applications were implemented: a simple mapping system (a kind of a small geographic information system) and an experimental medical expert system. Some concrete examples are described, within selected scenarios or analog ones. A
more detailed description of the applications is available with the code.

4.2. **Examples: a mapping system.**

The basic application domain selected is the mapping environment, where the need to integrate geographic and non-geographic data provides a reach variety of scenarios to test this approach. Within this environment, we can consider 3 types of *interface knowledge*: rules about the *presentation* of map data and objects, rules about which data is relevant for a given *user context*, and rules describing *domain knowledge* about typical map objects and map queries.

4.2.1. **Application description.**

Considering different types of map data (elevation, features), two tools were developed:

4.2.1.1. **Point Probe.**

This tool is intended to allow the user to query the map graphically (pointing with a tablet's stylus); all information available related with the specified point, such as coordinates, elevation, surface temperature, nature of feature (soil, river, road, etc), is displayed on the screen.

The issue here is to have the decisions on how to represent and display each type of data (values, legends, comments, pictures, etc) made by the interface, without the need to have it re-coded for each application program, or for each set and type of map data.

Naturally, these decisions are inferred also from the map data, but since they are defined by rules at the interface system, they can be influenced by other
factors (e.g., user preferences), and therefore can be changed at the interface level without affecting the content of the map files, and vice-versa.

For example, in my application I have one map representing elevation data, and another containing information that makes possible to calculate an estimate for the surface temperature. Both maps can be transformed in multi-layered maps, thanks to two translucent bitmaps with transparent background representing, respectively, water and road network.

Using the map probe tool, when I "click" on one of the maps, the application delivers a value, its units, and a request to the interface system to display it. It is the interface system that through its rules, identifies what type of interaction is in question, and selects the more adequate presentation-object to use.

If I configured the application profile for a "text-type" default presentation, the output is a string window, containing the value and units; but since the domain environment is mapping, it decides to display those string-windows directly on the map surface, and therefore they have to be "transparent-background" type. But if I then change the convenient rule to set instead a condition of "graphic-type" presentation, the same action (using the probe to "click" on either of the maps) will generate a very different output: a bar graph for the elevation value, and a thermometer for the temperature value.

It is important to notice that, since the choice of the particular data representation is not specified by the mapping application, the interface system is able to respond adequately to any other map, or any modification of data values or types within the same map. Conversely, changing the presentation-objects or the criteria for selection (interaction model) in the interface system, does not affect in any way the application code. Therefore, we obtained the desired level of separation between the semantic part of the
interaction, and the syntactic-lexical part of it.

![Diagram showing map data representation with interface rules]

**Fig.9**
Inferring map data representation with interface rules

4.2.1.2. Path study.

This tool is intended to allow the user to query the map also graphically, but this time along lines instead of points.

The user is allowed to draw any arbitrary path on a map, either a simple map with elevation data, either a multi-layered map with layers of road network, rivers, etc. Then, the tool checks sample points along this path, and draws a 2D profile of the elevation for the path, based on those points.

The issues here are not just a simple accumulation of the point probe's ones, since it implies 1) to correlate the punctual information along a path with a certain degree of consistency, 2) filter the type and amount of data to display, to avoid "data overdose" problems. The idea under scrutiny for future work is to include knowledge about "goal profiles" in the interface. A few examples:

- Path study for *road construction*. Relevant information can be: continuous elevation data, slope, land use, material surface.
- Path study for FM radio-transmissions. Relevant information can be: Z values for terminal locations, and all points in the line-of-sight that have greater Z value.

- Path study for flight plan. Relevant information can be: elevation data, max. Z on path, landing areas, height of trees and buildings if any, meteorological data.

4.2.2. Interface knowledge in the mapping application.

4.2.2.1. Inferring the environment.

The first thing the interface system does is to infer from available data the characteristics of the environment, and make some decisions on global presentation issues. For example, the interface is able to query by itself data such as: the machine name (host), the user name (login file), the application name (application load file). This provides the interface with an automatic configuration ability, able to adapt to different environments with different requirements and specifications (for instance, different screen resolution, information that is crucial for layout management and frame buffer size, information that is mandatory for bitmap formatting, color assignment, and other basic window system operations).

From there, a few rules containing the interface knowledge on "user-classification", "domain-classification", "resources-identification" and "presentation-definiers" are triggered and eventually fired by the knowledge-based interface-system:
RULES

(RULE CHECK-USER
   IF
       USER IS-NOT KNOWN
       USER IS-NOT CHECKED
   THEN
       ASK-OBJECT (KERNEL 'get-user-name *user-name*)
       DELETE-COND (USER IS-NOT CHECKED)
   )

(RULE LOAD-USER
   IF
       USER-NAME IS (>> U-name)
       USER IS-NOT LOADED
   THEN
       TELL-OBJECT (KERNEL 'load-user-param (<< U-name)
       DELETE-COND (USER IS-NOT LOADED)
   )

(RULE DEFAULT-USER
   IF
       USER-NAME IS-NOT KNOWN
   THEN
       ASSERT-COND (USER-BACKGROUND IS Computer-Expert)
       ASSERT-COND (USER-PRESENTATION-STYLE IS Text)
       ASSERT-COND (USER-LEVEL IS Expert)
       ASSERT-COND (USER-NATIONALITY IS Portugal)
       DELETE-COND (USER-NAME IS-NOT KNOWN)
   )

(RULE PEDRO-USER
   IF
       USER-NAME IS Pedro FA
   THEN
       ASSERT-COND (USER-BACKGROUND IS Computer-Expert)
       ASSERT-COND (USER-PRESENTATION-STYLE IS Text)
       ASSERT-COND (USER-LEVEL IS Expert)
       ASSERT-COND (USER-NATIONALITY IS Portugal)
       DELETE-COND (USER IS-NOT KNOWN)
   )
(RULE BOB-USER
   IF
      USER-NAME IS Bob Sab
   THEN
      ASSERT-COND (USER-BACKGROUND IS Computer-Expert)
      ASSERT-COND (USER-PRESENTATION-STYLE IS Iconic)
      ASSERT-COND (USER-LEVEL IS Expert)
      ASSERT-COND (USER-NATIONALITY IS USA)
      DELETE-COND (USER IS-NOT KNOWN)
)

(RULE MING-USER
   IF
      USER-NAME IS Ming
   THEN
      ASSERT-COND (USER-BACKGROUND IS Graphic-Design)
      ASSERT-COND (USER-PRESENTATION-STYLE IS Text)
      ASSERT-COND (USER-LEVEL IS Expert)
      ASSERT-COND (USER-NATIONALITY IS China)
      DELETE-COND (USER IS-NOT KNOWN)
)

(RULE ANNABELLE-USER
   IF
      USER-NAME IS Annabelle
   THEN
      ASSERT-COND (USER-BACKGROUND IS Civil-Expert)
      ASSERT-COND (USER-PRESENTATION-STYLE IS Text)
      ASSERT-COND (USER-LEVEL IS Average)
      ASSERT-COND (USER-NATIONALITY IS England)
      DELETE-COND (USER IS-NOT KNOWN)
)

(RULE CHECK-APPLICATION
   IF
      APPLICATION IS-NOT KNOWN
      APPLICATION IS-NOT CHECKED
   THEN
      ASK-OBJECT (KERNEL 'get-application-name *application-name*)
      DELETE-COND (APPLICATION IS-NOT CHECKED)
)

(RULE LOAD-APPLICATION
   IF
      APPLICATION IS-NOT LOADED
      APPLICATION-NAME IS (>> A-name)
   THEN
      TELL-OBJECT (KERNEL load-application-rules (<< A-name))
      ;; TELL-OBJECT (KERNEL load-application-param (<< A-name))
      DELETE-COND (APPLICATION IS-NOT LOADED)
      ASSERT-COND (APPLICATION IS LOADED)
)
(RULE DEFAULT-APPLICATION
  IF
    APPLICATION-NAME IS NOT KNOWN
  THEN
    ASSERT-COND (APPLICATION-DOMAIN IS Interface)
    ASSERT-COND (APPLICATION-PRESENTATION-TYPE IS GRAPHIC)
    DELETE-COND (APPLICATION-NAME IS NOT KNOWN)
)

(RULE MAP-DARPA-APPLICATION
  IF
    APPLICATION-NAME IS Map-Darpa
  THEN
    ASSERT-COND (APPLICATION-DOMAIN IS Map)
    ASSERT-COND (APPLICATION-PRESENTATION-TYPE IS TEXT)
    TELL-OBJECT (KERNEL 'appl-present 'text)
    DELETE-COND (APPLICATION-NAME IS NOT KNOWN)
)

(RULE WINDAD-APPLICATION
  IF
    APPLICATION-NAME IS Windad-ES
  THEN
    ASSERT-COND (APPLICATION-DOMAIN IS Map)
    ASSERT-COND (APPLICATION-PRESENTATION-TYPE IS NUMERIC)
    TELL-OBJECT (KERNEL 'appl-present 'numeric)
    DELETE-COND (APPLICATION-NAME IS NOT KNOWN)
)

(RULE MEDICAL-ES-APPLICATION
  IF
    APPLICATION-NAME IS Medical-ES
  THEN
    ASSERT-COND (APPLICATION-DOMAIN IS Medical)
    ASSERT-COND (APPLICATION-PRESENTATION-TYPE IS GRAPHIC)
    TELL-OBJECT (KERNEL 'appl-present 'graphic)
    DELETE-COND (APPLICATION-NAME IS NOT KNOWN)
)

(RULE CHECK-RESOURCES
  IF
    RESOURCES IS NOT KNOWN
    RESOURCES IS NOT CHECKED
  THEN
    ASK-OBJECT (KERNEL 'get-computer-name *computer-name*)
    DELETE-COND (RESOURCES IS NOT CHECKED)
)
(RULE DEFAULT-RESOURCES  
  IF  
    COMPUTER-NAME IS-NOT KNOWN  
  THEN  
    ASSERT-COND (MACHINE-TYPE IS Gator)  
    DELETE-COND (COMPUTER-NAME IS-NOT KNOWN)  
)

(RULE FRITZ-RESOURCES  
  IF  
    COMPUTER-NAME IS Fritz  
  THEN  
    ASSERT-COND (MACHINE-TYPE IS Renaissance)  
    DELETE-COND (RESOURCES IS-NOT KNOWN)  
)

(RULE MOUSER-RESOURCES  
  IF  
    COMPUTER-NAME IS Mouser  
  THEN  
    ASSERT-COND (MACHINE-TYPE IS Gator)  
    DELETE-COND (RESOURCES IS-NOT KNOWN)  
)

(RULE SCRAPPY-RESOURCES  
  IF  
    COMPUTER-NAME IS Scrappy  
  THEN  
    ASSERT-COND (MACHINE-TYPE IS Renaissance)  
    DELETE-COND (RESOURCES IS-NOT KNOWN)  
)

(RULE BASE-PRESENTATION  
  IF  
    MACHINE-TYPE IS Renaissance  
    APPLICATION-DOMAIN IS Interface  
  THEN  
    ASSERT-COND (ENVIRONMENT IS B-GRID)  
    TELL-OBJECT (SCREEN 'grid-step-default 150)  
    TELL-OBJECT (SCREEN 'grid-PREFIX "b")  
)
(RULE MAP-PRESENTATION
   IF
      MACHINE-TYPE IS Renaissance
      APPLICATION-DOMAIN IS Map
   THEN
      ASSERT-COND (ENVIRONMENT IS R-GRID)
      TELL-OBJECT (SCREEN 'grid-step-default 150)
      TELL-OBJECT (SCREEN 'grid-PREFIX "g")
)

(RULE MEDICAL-PRESENTATION
   IF
      MACHINE-TYPE IS Renaissance
      APPLICATION-DOMAIN IS Medical
   THEN
      ASSERT-COND (ENVIRONMENT IS M-GRID)
      TELL-OBJECT (SCREEN 'grid-step-default 150)
      TELL-OBJECT (SCREEN 'grid-PREFIX "m")
)

(RULE GATOR-PRESENTATION
   IF
      MACHINE-TYPE IS Gator
      APPLICATION-DOMAIN IS Interface
   THEN
      ASSERT-COND (ENVIRONMENT IS Gator-GRID)
      TELL-OBJECT (SCREEN 'grid-step-default 140)
      TELL-OBJECT (SCREEN 'grid-PREFIX "i")
)

(RULE BASE-PARAM-PRESENTATION
   IF
      ENVIRONMENT IS B-GRID
   THEN
      TELL-OBJECT (SCREEN 'font "latin.24x8")
      TELL-OBJECT (SCREEN 'text-color '(0 0 0))
      TELL-OBJECT (SCREEN 'background-color '(180 210 200))
      TELL-OBJECT (SCREEN 'background-border-color '(150 150 160))
      TELL-OBJECT (SCREEN 'background-border 0)
      TELL-OBJECT (SCREEN 'text-highlight-color '(200 20 10))
      DELETE-COND (ENVIRONMENT IS B-GRID)
)
(RULE MAP-PARAM-PRESENTATION-1
   IF
     ENVIRONMENT IS R-GRID
     USER-BACKGROUND IS Computer-Expert
   THEN
     TELL-OBJECT (SCREEN 'font "latin.24x8")
     TELL-OBJECT (SCREEN 'text-color '(0 0 0))
     TELL-OBJECT (SCREEN 'background-color '(130 130 140))
     TELL-OBJECT (SCREEN 'background-border-color '(150 150 160))
     TELL-OBJECT (SCREEN 'background-border 1)
     TELL-OBJECT (SCREEN 'text-hlight-color '(200 20 10))
   DELETE-COND (ENVIRONMENT IS R-GRID)
)

(RULE MAP-PARAM-PRESENTATION-2
   IF
     ENVIRONMENT IS R-GRID
     USER-BACKGROUND IS Graphic-Design
   THEN
     TELL-OBJECT (SCREEN 'font "latin.24x8")
     TELL-OBJECT (SCREEN 'text-color '(200 200 200))
     TELL-OBJECT (SCREEN 'background-color '(0 0 0))
     TELL-OBJECT (SCREEN 'background-border-color '(150 150 160))
     TELL-OBJECT (SCREEN 'background-border 1)
     TELL-OBJECT (SCREEN 'text-hlight-color '(20 20 100))
   DELETE-COND (ENVIRONMENT IS R-GRID)
)

(RULE MEDICAL-PARAM-PRESENTATION
   IF
     ENVIRONMENT IS M-GRID
     USER-BACKGROUND IS Computer-Expert
   THEN
     TELL-OBJECT (SCREEN 'font "latin.24x8")
     TELL-OBJECT (SCREEN 'text-color '(0 0 0))
     TELL-OBJECT (SCREEN 'background-color '(200 200 240))
     TELL-OBJECT (SCREEN 'background-border-color '(150 150 160))
     TELL-OBJECT (SCREEN 'background-border 0)
     TELL-OBJECT (SCREEN 'text-hlight-color '(200 20 10))
   DELETE-COND (ENVIRONMENT IS M-GRID)
)

(RULE LEGEND-PRESENTATION-1
   IF
     USER-LEVEL IS Average
     USER-PRESENTATION-STYLE IS Iconic
   THEN
     ASSERT-COND (LEGEND-LEVEL IS Help)
)

44
(RULE LEGEND-PRESENTATION-2
  IF
    USER-LEVEL IS Expert
    USER-PRESENTATION-STYLE IS Text
  THEN
    ASSERT-COND (LEGEND-LEVEL IS Functional)
)

(RULE LEGEND-PRESENTATION-3
  IF
    USER-LEVEL IS Expert
    USER-PRESENTATION-STYLE IS iconic
  THEN
    ASSERT-COND (LEGEND-LEVEL IS Regular)
)

(RULE NATIONAL-PRESENTATION-1
  IF
    USER-NATIONALITY IS USA
  THEN
    TELL-OBJECT (GWS 'unit-system 'non-metrical)
)

(RULE NATIONAL-PRESENTATION-2
  IF
    USER-NATIONALITY IS England
  THEN
    TELL-OBJECT (GWS 'unit-system 'non-metrical)
)

4.2.2.2. Creating Dialogue-Objects.

Next, we want to build the interface at a higher level than the graphic window system calls. For this, the system has a meta-comment Create-dialogue-Object, that allows the user to create on-the-fly an interface object according to its role in the computer-human interaction; or then, the system contains rules that identify the more adequate role for an interaction request coming from the application. These roles are in fact correspondent to the semantic level of the interface, representing types or chunks of dialogue.
Examples of *roles*:

Command, Meta-command, Cursor, Tag, Button-link, Display-picture, Display-number, Display-temperature, Display-quantity-1D, Display-quantity-2D, Display-Quantity-nD, Display-boolean, Display-text-file, Legend, Message, Query-text, Query-number, Query-boolean, Query-picture, Query-color.

Then, the system uses Goal-handler and Object-classifier rules, together with previous inferred conditions, to establish which adequate graphic calls are to be made to the graphic window system, in order to create the new interface object:

(RULE QUERY-TEXT-ROLE
 IF
  ASK (>> QUERY) (>> UNITS))
 THEN
  DELETE-COND (ASK (<< QUERY) (<< UNITS))
  ASSERT-COND ((<< QUERY) UNITS IS (<< UNITS))
  ASSERT-COND ((<< QUERY) ROLE IS QUERY-TEXT)
 )

(RULE QUERY-BOOLEAN-ROLE
 IF
  ((>> QUERY-R) UNITS IS YES/NO)
  ((>> QUERY-R) ROLE IS QUERY-TEXT)
 THEN
  DELETE-COND ((<< QUERY-R) UNITS IS YES/NO)
  DELETE-COND ((<< QUERY-R) ROLE IS QUERY-TEXT)
  ASSERT-COND ((<< QUERY-R) ROLE IS QUERY-BOOLEAN)
 )

(RULE DISPLAY-TEXT-ROLE
 IF
  TELL (>> MESSAGE-T) (>> VALUE-T) (>> UNITS-T))
 THEN
  ASSERT-COND ((<< MESSAGE-T) UNITS IS (<< UNITS-T))
  ASSERT-COND ((<< MESSAGE-T) VALUE IS (<< VALUE-T))
  ASSERT-COND ((COND ((<< MESSAGE-T) ROLE IS DISPLAY-TEXT))
  DELETE-COND (TELL (<< MESSAGE-T) (<< VALUE-T) (<< UNITS-T))
 )
(RULE DISPLAY-BOOLEAN-ROLE
   IF
     ((>> MESSAGE-B) UNITS IS YES/NO)
     ((>> MESSAGE-B) ROLE IS DISPLAY-TEXT)
   THEN
     ASSERT-COND ((<< MESSAGE-B) ROLE IS DISPLAY-BOOLEAN)
     DELETE-COND ((<< MESSAGE-B) UNITS IS YES/NO)
     DELETE-COND ((COND ((<< MESSAGE-B) ROLE IS DISPLAY-TEXT))
)

(RULE DISPLAY-NUMBER-ROLE
   IF
     ((>> MESSAGE-N) UNITS IS "feet")
     ((>> MESSAGE-N) ROLE IS DISPLAY-TEXT)
   THEN
     ASSERT-COND ((<< MESSAGE-N) ROLE IS DISPLAY-NUMBER)
     DELETE-COND ((<< MESSAGE-N) ROLE IS DISPLAY-TEXT)
)

(RULE DISPLAY-TEMPERATURE-ROLE
   IF
     ((>> MESSAGE-TP) UNITS IS "°F")
     ((>> MESSAGE-TP) ROLE IS DISPLAY-TEXT)
   THEN
     ASSERT-COND ((<< MESSAGE-TP) ROLE IS DISPLAY-TEMPERATURE)
     DELETE-COND ((<< MESSAGE-TP) ROLE IS DISPLAY-TEXT)
)

(RULE NAME-COMMAND-OBJECT
   IF
     ROLE IS Command
     USER-PRESENTATION-STYLE IS Text
     LEGEND-LEVEL IS Functional
   THEN
     ASSERT-COND (INTERFACE-OBJECT IS 'name-command)
     DELETE-COND (ROLE IS COMMAND)
)

(RULE LEGEND-COMMAND-OBJECT
   IF
     ROLE IS Command
     USER-PRESENTATION-STYLE IS Text
     LEGEND-LEVEL IS Help
   THEN
     ASSERT-COND (INTERFACE-OBJECT IS 'legend-command)
     DELETE-COND (ROLE IS COMMAND)
)
(RULE 8BIT-ICON-OBJECT
   IF
      ROLE IS Command
      USER-PRESENTATION-STYLE IS Iconic
      MACHINE-TYPE IS Gator
   THEN
      ASSERT-COND (INTERFACE-OBJECT IS '8bit-icon)
      DELETE-COND (ROLE IS COMMAND)
   )

(RULE 24BITMAP-OBJECT
   IF
      ROLE IS Display-picture
      MACHINE-TYPE IS Renaissance
   THEN
      ASSERT-COND (INTERFACE-OBJECT IS '24-bitmap)
      DELETE-COND (ROLE Display-picture)
   )

(RULE BAR-GRAPH-OBJECT
   IF
      ROLE IS Display-Quantity-2D
   THEN
      ASSERT-COND (INTERFACE-OBJECT IS 'bar-graph-display)
      DELETE-COND (ROLE Display-Quantity-2D)
   )

(RULE QUERY-TEXT-OBJECT
   IF
      ((QUERY-TX) ROLE IS QUERY-TEXT
   THEN
      ASK-USER ((<< QUERY-TX) 'answer*)
      DELETE-COND ((<< QUERY-TX) ROLE IS QUERY-TEXT)
   )

(RULE QUERY-BOOLEAN-OBJECT
   IF
      ((QUERY-BO) ROLE IS QUERY-BOOLEAN)
   THEN
      TELL-OBJECT (GWS CREATE-LIST-QUERY-BOOLEAN ((<< QUERY-BO)))
      DELETE-COND ((<< QUERY-BO) ROLE IS QUERY-BOOLEAN)
   )

(RULE DISPLAY-BOOLEAN-OBJECT
   IF
      ((DISPLAY-BO) ROLE IS DISPLAY-BOOLEAN)
      ((DISPLAY-BO) VALUE IS (VALUE-BO))
   THEN
      TELL-OBJECT (GWS CREATE-BOOLEAN-DISPLAY ((DISPLAY-BO) ((VALUE-BO))))
      DELETE-COND ((COND ((DISPLAY-BO) ROLE IS DISPLAY-BOOLEAN)
                  ((VALUE-BO) VALUE IS (VALUE-BO)))
   )

48
(RULE DISPLAY-TEXT-OBJECT
  IF
    ((>> DISPLAY-TX) ROLE IS DISPLAY-TEXT)
    ((>> DISPLAY-TX) UNITS IS "string")
    ((>> DISPLAY-TX) VALUE IS (>> VALUE-TX))
  THEN
    TELL-OBJECT (GWS CREATE-VALUE-DISPLAY ((<< DISPLAY-TX) "b25") " " (<< VALUE-TX)))
    DELETE-COND ((COND ((<< DISPLAY-TX) ROLE IS DISPLAY-TEXT))
    DELETE-COND ((COND ((<< DISPLAY-TX) VALUE IS (<< VALUE-TX)))
    DELETE-COND ((COND ((<< DISPLAY-TX) UNITS IS "string"))
)

(RULE DISPLAY-NUMBER-OBJECT
  IF
    ((>> DISPLAY-TX) ROLE IS DISPLAY-NUMBER)
    ((>> DISPLAY-TX) UNITS IS "feet")
    ((>> DISPLAY-TX) VALUE IS (>> VALUE-TX))
    APPLICATION-PRESENTATION-TYPE IS TEXT
  THEN
    TELL-OBJECT (GWS CREATE-VALUE-DISPLAY ((<< DISPLAY-TX) "b25") " " (<< VALUE-TX)))
    DELETE-COND ((COND ((<< DISPLAY-TX) ROLE IS DISPLAY-NUMBER))
    DELETE-COND ((COND ((<< DISPLAY-TX) VALUE IS (<< VALUE-TX)))
    DELETE-COND ((COND ((<< DISPLAY-TX) UNITS IS "feet"))
)

(RULE DISPLAY-BAR-OBJECT
  IF
    ((>> DISPLAY-N) ROLE IS DISPLAY-NUMBER)
    ((>> DISPLAY-N) UNITS IS "feet")
    ((>> DISPLAY-N) VALUE IS (>> VALUE-N))
    APPLICATION-PRESENTATION-TYPE IS GRAPHIC
  THEN
    TELL-OBJECT (GWS CREATE-BAR-DISPLAY ((<< DISPLAY-N) "m35" 0 1000 "feet" (<< VALUE-N)))
    DELETE-COND ((COND ((<< DISPLAY-N) ROLE IS DISPLAY-NUMBER))
    DELETE-COND ((COND ((<< DISPLAY-N) VALUE IS (<< VALUE-N)))
    DELETE-COND ((COND ((<< DISPLAY-N) UNITS IS "feet"))
)

(RULE DISPLAY-WEA-TERMOMETER-OBJECT
  IF
    ((>> DISPLAY-NTP) ROLE IS DISPLAY-TEMPERATURE)
    ((>> DISPLAY-NTP) UNITS IS "oF")
    ((>> DISPLAY-NTP) VALUE IS (>> VALUE-NTP))
    APPLICATION-PRESENTATION-TYPE IS GRAPHIC
  THEN
    TELL-OBJECT (GWS CREATE-TERMOMETER ((<< DISPLAY-NTP) "m35" nil nil "oF" (<< VALUE-NTP)))
    DELETE-COND ((COND ((<< DISPLAY-NTP) ROLE IS DISPLAY-TEMPERATURE))
    DELETE-COND ((COND ((<< DISPLAY-NTP) VALUE IS (<< VALUE-NTP)))
    DELETE-COND ((COND ((<< DISPLAY-NTP) UNITS IS "oF"))
)
3.2.2.3. Inferring screen layout.

But before the actual graphic window system calls can be executed, the interface system must infer, or query, other parameters. Among those are the layout data. The system handles the screen management through a package of layout
management routines, an inference-object SCREEN, and a set of Screen rules.

These rules can be bypassed, in which case each presentation-object will be positioned according to its own default values.

Example of screen layout management routines: Check-if-slot-free, Mark-slot-free, Mark-slot-full (complete set of routines in annex). The object SCREEN has all those routines as its methods, that can be activated through the firing of rules:

(RULE LAYOUT-1
  IF
  Environment IS R-GRID
  Grid-step IS 150
  Interface-object IS 'name-command
  THEN
  Assert-cond (Default-slot IS "g63")
)

(RULE LAYOUT-2
  IF
  Default-slot IS (>> slot)
  Then
  Ask-object (SCREEN ‘check-if-free (<< slot))
  Assert-cond (SCREEN ‘check-if-free (result-value))
  Ask-object ( (<< slot) ‘current-font )
  Assert-cond ( (<< slot) ‘current-font IS (result-value))
)

(RULE LAYOUT-3
  IF
  Default-slot IS (>> slot)
  SCREEN ‘check-if-free IS NIL
  User-presentation-style IS Text
  Role IS (>> role)
  THEN
  Delete-cond (SCREEN ‘check-if-free IS NIL)
  Assert-cond ( SHRINK (<< role) FONT)
(RULE LAYOUT-4
  IF
    SHRINK (>> role) FONT
    (>> slot) 'current-font IS "latin.24x8"
  THEN
    TELL-OBJECT ( (<< slot) 'CHANGE-FONT ("LATIN.16X8") )
    DELETE-COND ( (<< slot) 'current-font IS "latin.24x8" )
    ASSERT-COND ( (<< slot) 'current-font IS "latin.16x8" )
    DELETE-COND (SHRINK (<< role) FONT )
)

(RULE LAYOUT-5
  IF
    SHRINK (>> role) FONT
    (>> slot) 'current-font IS "latin.16x8") ; minimo
  THEN
    ASK-OBJECT ( SCREEN 'get-free-slot (<< role) (<< slot) )
    ASSERT-COND ( SCREEN 'get-free-slot (result-value) )
    TELL-OBJECT ( (<< slot) 'CHANGE-FONT ("LATIN.24X8") )
    DELETE-COND ( (<< slot) 'current-font IS "latin.16x8" )
    ASSERT-COND ( (<< slot) 'current-font IS "latin.24x8" )
    DELETE-COND (SHRINK (<< role) FONT )
    DELETE-COND (DEFAULT-SLOT IS (<< slot )
)

(RULE LAYOUT-6
  IF
    SCREEN 'get-free-slot IS (>> slot)
    ROLE IS Command
  THEN
    ASK-OBJECT ( (<< slot) 'get-y )
    ASSERT-COND ( (<< slot) 'get-y IS (result-value) )
)

(RULE GWS-1
  IF
    INTERFACE-OBJECT IS 'name-command
  THEN
    ASK-USER ("Do Function?" DoFunc symbol)
    ASSERT-COND (DOFUNC IS DoFunc)
    ASK-USER ("Name?" Name string)
    ASSERT-COND (NAME IS Name)
)
These rules are a sample of the interface knowledge used; the more exhaustive list is available, with the code.

The rule-based structure of the interface system brings us enormous advantages over traditional programming. On one hand, it creates a desirable encapsulation of the graphic routines; the user can deal with the interface at the most high level. On the other hand, it brings a flexibility hard to match: the user can adapt easily (and on-the-fly) the interface to his own needs and preferences, just by adding, deleting, or changing rules. The program (inference engine, support packages) doesn't have to be modified for that.
4.2.4. **Learning in the mapping application.**

Once that the knowledge about these "user goal profiles" is represented in the interface, generalization and specialization heuristics will allow the system to infer new rules to respond to new similar-but-not-the-same goals, and make clever suggestions. Although currently not implemented, the design and philosophy of the inference engine implemented is prepared to support this future extension, as explained in next Chapter. A few examples:

- Path study for *road*. Interface system doesn't know about it, but finds best matching rule: "Path study for road construction", drops condition "construction", and generates new rule by analogy (generalization).

- Path study for *helicopter flight plan*. The same as above happens, this time the interface system chooses "flight plan" and adds new condition - "for helicopter" - generating new rule by analogy (specialization).

Naturally, the user can introduce some fine tuning in these new profiles, but the task is easier and the consistency with previous rules is kept.
4.3. **Examples: a medical expert system**

Fig 10  
Medical expert system with "traditional" graphic interface structure

In the first chapter, it was introduced a simple medical expert system as an example for the type of problems brought by a changing environment into interface design. In general, we could conclude that this system has a component which is easy to modify - the medical knowledge - and another one which is more painful to change or extend - the graphical presentation of the input/output.

But a knowledge-based graphical interface can change this scenario. One of the main reasons is that, now, the graphical presentation of the input/output and its control, is separated from the graphical interpreter and represented in the knowledge base:
Medical expert-system with a rule-based graphical interface

This can be achieved by representing the interaction elements (such as boxes, thermometers, etc) as frames, and their control conditions as rules, as exposed above.

Example of a frame:

Frame CROSS-BOX:
   class: IS-A Qualitative [meta-class: Qualitative IS-A Dialogue-Object]
   attributes:
      shape (default: square)
      size (default: 20)
      location (X, Y)
      color (default: black)
      legend (default type: text)
   input method: read cross character on stylus click
   output method: draw itself (arg.: attributes)

End Frame

Other examples of frames: Color-box, Thermometer, Gauge, Bar, Numeric-box, Graph, etc. In qualitative objects, selection would be among color-box, cross-box, icon, etc; in quantitative objects, selection would be among gauge, thermometer, bar, numeric-box, graph, etc.
In the case of the referred medical Expert-System, the application itself is a rule-based system. A very interesting side effect of this interface philosophy, is that applications that are themselves rule-based systems can share the same inference engine, the same knowledge-base shell, with the interface system.

Essentially, what we have is a different set of rules for the domain knowledge, in this case, medical knowledge.

Example of medical rules:

(RULE MEDIC-1
   IF
     (BODY-AREA IS NOSE)
   THEN
     ASSERT-COND (ASK (ITCHING? SNEEZING? RUNNING-NOSE? COUGHING?) YES/NO)
     DELETE-COND (BODY-AREA IS NOSE)
   )

(RULE DIAGNOSIS-ALEGRYMEDIC-1
   IF
     (ITCHING? YES)
     (RUNNING-NOSE? YES)
     (SNEEZING? NO)
     (COUGHING? NO)
   THEN
     ASSERT-COND (TELL "Diagnosis-> " "Mild Alergy" "string")
     DELETE-COND (ITCHING? YES)
     DELETE-COND (COUGHING?NO)
     DELETE-COND (SNEEZING? NO)
     DELETE-COND (RUNNING-NOSE? YES)
   )
(RULE DIAGNOSIS-ALERGY-MEDIC-2
 IF
  (ITCHING? YES)
  (RUNNING-NOSE? YES)
  (SNEEZING? YES)
  (COUGHING? NO)
 THEN
  ASSERT-COND (TELL "Strong Alergy. Fever-> " NO YES/NO)
  DELETE-COND (ITCHING? YES)
  DELETE-COND (COUGHING?NO)
  DELETE-COND (SNEEZING? YES)
  DELETE-COND (RUNNING-NOSE? YES)
)

(RULE DIAGNOSIS-FLU-MEDIC
 IF
  (ITCHING? YES)
  (RUNNING-NOSE? YES)
  (SNEEZING? YES)
  (COUGHING? YES)
 THEN
  ASSERT-COND (TELL "Flu. Fever Expected-> " YES YES/NO)
  DELETE-COND (ITCHING? YES)
  DELETE-COND (COUGHING? YES)
  DELETE-COND (SNEEZING? YES)
  DELETE-COND (RUNNING-NOSE? YES)
)

In my experimental Expert System, the user "clicks" on a bitmap representing
the human body, in order to indicate which is the area of medical concern. A
rule containing medical knowledge will be fired, activating a series of questions
to the user/patient; if these questions are yes/no questions, the interface system
will infer that the type of dialogue is "Query-Boolean", and then, whether the
presentation-type is graphic or textual, it will select either a get-string-window,
or a get-bbolean-window as the adequate objects.

The answer is inferred from the medical-knowledge rules; but here again those
rules just ask the interface to display a value and its units. It is up to the interface to find the dialogue-object that is going to proceed with the interaction.

If I want to change a medical rule, for instance adding another question to the rule body-area-medic-1, I will not need to introduce any change in the interface system or its code; the system easily generates another convenient dialogue-object. And if again the medical knowledge is changed, for instance giving a numeric value for the expected fever instead of a yes/no answer, the system will be able to select the thermometer (created for the mapping application), instead of a boolean box or a string, without the need of any output specification in the medical rules.

But my previous thermometer was a kind of a weather thermometer; it displays a white bar if the value is below freezing point (32 F), green if its mild (>0 and < 100), red if too hot, etc.

If now I want instead to change the interface, for instance, creating a special presentation of the object "thermometer" that represents the values of temperature according to a medical criteria instead of a weather criteria (displaying a green bar for no fever (> 98 <99), and a red bar for fever (>= 99), etc), I can do it simply by adding the new presentation-object to my object-base, and the role "display-body-temperature" to my interaction model. Then, I can specify that the role "display-temperature" is the default (or valid for map-domain type of applications), but if the application-domain is medical, then it is expected that the default role for display-temperature is display-body-temperature, with the correspondent special thermometer object.

And no part of the application had to be touched, in this case, none of the medical rules, during the process.
4.4. **Examples: Conclusion**

The power of this approach can then be emphasised by the following facts:

a) We can use a great part (if not all) of this Schemata and Inference Net, for other applications;

b) Applications don't need to be rule-based systems (like the medical expert system). The referred Schemata and Inference Net can be the only content of the rule-based system, or more correctly, of the *knowledge-based system*, since we have knowledge represented in rules and frames.

This directly supports the notion that interface *knowledge* as an entity can exist separately from any particular application, yet be shared by several.
5. The tools

5.1. The first options.

As a first option, the graphical component would use VLW - *Badwindows* [Greenlee 87] and Starbase graphic software (HP - Bobcats); the Knowledge-Base would use *HPRL*, both sharing a Common Lisp environment.

*HPRL* stands for "Heuristic Programming and Representation Language", and was introduced as a language for building expert systems [Rosemberg 82]. Basically, it provides: A network of frames organized as a partially ordered hierarchy permitting multiple inheritance; Slots on frames whose values may be other frames, permitting multiple relations among frames; Procedural attachements to frames; Rule-based inferencing (both backward chaining and forward chaining), and Agenda control.

The choice of the tools, and mainly of HPRL, was in great part consequence of practical aspects such as: a) It was the only integrated programming environment (graphical interpreter - knowledge base shell ) available at the moment; b) It was at least partially tested.

The main problems with HPRL were: speed (too slow), built-in interface (no real graphic interface, too heavy), and knowledge storage (no simple mechanisms provided).

But maybe the worst problem was the inherent risk associated with using an experimental tool, since HP no longer supports it. That risk was extended to the future non-portability of my code, since other components of the system are
being upgraded. Right now, for instance, HPRL does not run with the new Lucid Lisp; this means that to use HPRL, an old version of lisp would have to be kept on the system.

Possible alternatives studied were:

For Graphic window systems: X11 (instead of Badwindows);
For KBS: KEE [Kehler 84], or to write myself a dedicated small inference engine.

Once it became clear that I would not have KEE available (on the HP Bobcats), the decision made was to build my self an inference engine, fully compatible with the VLW graphic window system, "Badwindows". This represented a substantial part of the code effort, as it would be expectable, and was time consuming. The major advantage is the fact that it will be a tool integrated in the graphic programming environment, available for any other knowledge-based graphic applications to be developped in the future in the Laboratory.

5.2. The inference engine.

5.2.1. General description.

Given the nature of the knowledge-based application it was ment to serve - Graphic Interface - the algorithm implemented was basically a forward-chaining inference engine. But its structure was designed in such way that it won't be too hard to add in the future backward-chaining reasoning; the representation of the conditional expressions is uniform, either in the IF part, either in the THEN part.
According to the described model for the knowledge-based graphic interface system, the inference-engine supports a set of operations that allow the rules to interact with the Working Memory (present conditions of the world and relevant history), and with a Object-Data-Base. There are two types of objects in the Object data-base: Dialogue-objects, and Inference-objects. Dialogue-objects are all those that are protagonists in some direct interaction with the user (input/output); Inference-objects are essentially internal data-structures of the inference-engine.

Also according with the described model of presentation-objects, each dialogue-object has a dual facet: a window-facet (presentation-object) and a symbol-facet (object itself). These facets are kept in harmony through graphic and symbolic support packages, but are perfectly distinct; in fact, the window-facet is managed by the graphic interpreter, while the symbolic-facet is managed by the master program of the interface.

5.2.2. Structure.

The knowledge-base package include:

- The Knowledge-Base inference-engine-fc (forward-chaining) program;

- The set of Knowledge-Base operations (ASSERT-COND, DELETE-COND, ASSERT-RULE, DELETE-RULE, ASSERT-ACTION, DELETE-ACTION, ASK-USER, TELL-USER, ASK-OBJECT, TELL-OBJECT, EVAL);

- The Knowledge-Base Management Functions for Conditions;
- The Knowledge-Base Management Functions for Actions;
- The Knowledge-Base Management Functions for Rules;
- The Knowledge-Base Class Rules Files:
(user-classifiers, domain-classifiers, domain-constraints, resources-selectors, screen-handlers, user-goal-definers, user-goal-handlers, menu-handlers, dialogue-type-classifiers, object-type-classifiers, object-type-selectors, presentation-definers presentation-selectors, method-selectors, no-classified).

5.2.3. Knowledge representation.

The knowledge-base is essentially a rule-based system. Because the inference engine matcher is able to handle pattern variables, rules can contain variables in either IF or THEN part; the values of these variables can be binded at run-time, providing a link between the conditions and the actions of the rule, and also creating the adequate framework for generalization/specialization heuristics.

The knowledge representation has the following formats:

Rule format:

(RULE ( (Name rule-name) (Class rule-class) (N-cond n-of-conds)
     (Cond-matches n-of-cond-matches)
     ( (Cond1) (Cond2) (Cond3)...... (Condn) ............... )
     ( (Action1) (Action2) (Action3) ...... (Actionk) ... )
    )

)

64
Condition format:

(COND (expression relation expression)
   (Class rule-class1 rule-class2 rule-class3 ... / all)
   (Weight cond-weight)
   (Match-rule rule1 rule2 ... / none)
)

Action format:

(ACTION operation (argument-list))

Operation formats:

ASSERT-COND (cond &opt rule-name)
DELETE-COND (cond &opt rule-name)
ASSERT-ACTION (action rule-name)
DELETE-ACTION (action rule-name)
ASSERT-RULE (rule)
DELETE-RULE (rule)
ASK-USER (message variable var-type &opt var-output)
TELL-USER (message)
ASK-OBJECT (object object-var/method &opt var-output)
TELL-OBJECT (object object-var/method param-list)

Argument-list format:

(expression expression ...)

Expression format:

atom | list | atom-pattern | atom-pattern-sequence | pull-pattern-variable | push-pattern-variable
5.2.4 Inference engine algorithm.

The complete algorithm is included in annex (with the code). Here follows a brief description:

BEGIN INFERENCE;

Initialize-inference-structures;
Read Rule-files;
Read Input-Conditions;

UNTIL end-of-rule-classes
    Get-next-rule-class;
    
    UNTIL end-of-conditions
        Get-next-condition;
        IF condition IS of current-rule-class
            THEN
                Check if match some rule;
                IF match THEN Update-inference-structures; Get-next-condition;
                ELSE Get-next-condition
            ELSE Get-next-condition
        END-UNTIL end-of-conditions
    END-UNTIL end-of-conditions

UNTIL end-of-triggered-rule-list
    Resolve-collisions;
    Execute-rule;
    Update-inference-structures;
    IF Action-Operations IS ASSERT/DELETE-COND
        THEN Repeat-same-class
        ELSE Get-next-rule-class
    END-UNTIL end-of-triggered-rule-list

    IF fired-rule-list IS empty AND rule-class IS mandatory
        THEN Try-generation-new-rule; Repeat-triggered-list-cycle;
        ELSE Get-next-rule-class
    END-UNTIL end-of-rule-classes

Restore-inference-structures

END INFERENCE.
6. **Code components and programming environment**

The main code components are:

Inference engine; knowledge representation - rules and frames; meta-knowledge representation - goals and plans; bibliographic data-base functions; mapping manipulator functions; presentation objects - window archetypes, icon library; knowledge base storage and retrieval; interface meta-command functions; mini-graphic translator for the knowledge representation.

This project was implemented on HP graphic workstations, with Renaissance frame buffer, and is written in HP Object-oriented Common Lisp (Lucid Lisp).
Acknowledgements

A special mention is due to my sponsors, that made possible the completion of my Master of Science degree at MIT:

- Comissão Cultural Luso-Americana, FULBRIGHT Grant
- Laboratorio Nacional de Engenharia e Tecnologia Industrial
- Fundação Luso-Americana para o Desenvolvimento
- MIT Media Laboratory - Visible Language Workshop

This project would also not have be possible without the help, criticism and guidance of Muriel Cooper, my thesis adviser; Patrick Purcell and Duvvuru Sriram, my thesis readers; Henry Lieberman and Ron MacNeil, my informal thesis co-advisers; and the friendly environment provided by those working at VLW, in particular my graduate students colleagues Sylvain Morgaine, Suguru Ishizaki, David Small, Laura Robin, Ming Chen and Russell Greenlee.

Thanks are due to all of them.
**Glossary**

- **Context understanding**: "(...) a system that not only responds to you on the basis of explicit enquiries via combinations of speech, touch, and gesture, but one that reacts automatically and subtly to cues and clues implicit in the way you look at it. Two kinds of knowledge would be vital to such a system: What is the user interested in? And how well is the user following the exposition of the topic?" [Bolt 84].

- **Direct manipulation**: "visibility of the object of interest; rapid, reversible, incremental actions; and replacement of complex command language syntax by direct manipulation of the object of interest" [Shneiderman 83];

- **Event-Response systems**: Rule-based interface interpreters; events are modelized through boolean flags, and represented in IF-THEN rules. See [Hill 87].

- **Frames**: "A frame is a data-structure for representing a stereotyped situation...Attached to each frame are several kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed" [Minsky 74]

- **Presentation systems**: "model of user interfaces based on connecting a descriptive data base describing the application domain to a presentation data base modelling the screen" [Lieberman 85]. See [Zdybel 82], [Cicarelli 84], [Szekely 87].

- **Programming by example**: "Do what I mean" programming [Myers 86]: the user gives a number of examples and the system tries to infer the general
program structure;

- **Programming by rehearsal**: "Do what I did" programming: the user works on an example, and the system executes normally, but memorizing for later use [Lieberman 86];

- **Program visualization**: Graphical display of textually specified code and/or data [Lieberman 84];

- **Visual Programming**: "any system that allows the user to specify a program in a two (or more) dimensional fashion" [Myers 86].
References

[Anderson 86]  

[Baecker 80]  

[Barnard 87]  

[Bauer 78]  

[Bolt 84]  

[Borning 86]  

[Card 80]  

[Cicarelli 84]  

[Davis 77]  

[Feldman 82]  

[Ferraz de Abreu 87]  
[Ferraz de Abreu 88]

[Foley 88]

[Greenlee 87]

[Hill 87]

[Kasic 82]

[Kay 85]

[Kehler 84]

[Lieberman 82]

[Lieberman 84]

[Lieberman 85]

[Lieberman 86]
[Lieberman 88]  

[Mackinlay 86]  

[Minsky 74]  

[Monteiro 81]  

[Myers 86]  

[Myers 87]  

[Olsen 84]  

[Poltrock 86]  

[Quinn 86]  

[Rich 83]  

[Rissland 84]  
[Roach 82]

[Rosemberg 82]

[Schneiderman 83]

[Stefik 84]

[Szekely 87]

[Tanner 83]

[Tyler 88]

[Zdybel 82]