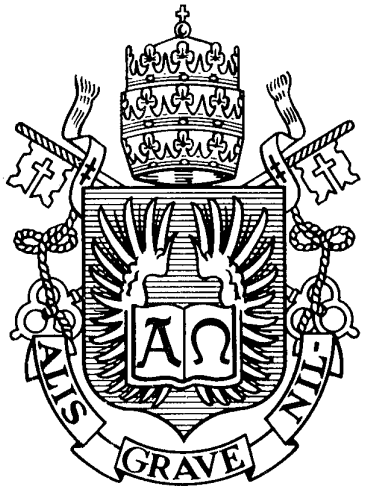

PUC

Série : Monografias em Ciência da Computação
No. 15/88

PUC : A KNOWLEDGE BASED ENVIRONMENT FOR PLANNED USER COMMUNICATION

Paula Y. Guarangys
Carlos J. P. Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

IA MARQUÊS DE SÃO VICENTE, 225 – CEP 22453

RIO DE JANEIRO – BRASIL

PUC/RJ - Departamento de Informática

Série : Monografias em Ciência da Computação, 15/88
Editor : Paulo A. S. Veloso Outubro, 1988

PUC : A KNOWLEDGE BASED ENVIRONNEMT FOR PLANNED USER COMMUNICATION *

Paula Y. Guarangys
Carlos J. P. Lucena

* This work has been partially supported by CNPq and SID Informática, Brazil.

Para obter cópias :
Rosane T. L. Castilho
Assessoria de Biblioteca, Documentação e Informação
Rua Marquês de São Vicente, 225 - Gávea
22.453 - Rio de Janeiro, RJ.
Brasil

PUC: A KNOWLEDGE BASED ENVIRONMENT FOR PLANNED USER COMMUNICATION

Paula Y. Guaranys*
Carlos J. P. Lucena
Departamento de Informática
Pontifícia Universidade Católica
Rio de Janeiro - Brazil

Abstract

PUC's requirements are based on our view that the ideal environment for designing a personalised interface is one which involves the collaboration of the user and an interface designer who knows about the application. The approach used in the design of PUC was to incorporate the interface designer "into the environment" and invite the user to produce his/her own interface with its assistance. The approach aims to allow the end user to plan his/her future interaction with an application while learning about it.

number of words: approx. 3000
key words: interface design, user plan, knowledge based environment, interaction style
subject area: intelligent interfaces, software engineering

* This work was partially supported by CNPq and SID Informática, Brazil

PUC: A KNOWLEDGE BASED ENVIRONMENT FOR PLANNED USER COMMUNICATION

Paula Y.Guaranys
Carlos J.P.Lucena
Departamento de Informática
Pontifícia Universidade Católica
Rio de Janeiro - Brazil

1. Introduction

The general problem of making computers easier to use is a major enterprise of obvious importance. We consider here a user interface as any computer software that has as its primary function the task of providing the user with information that will assist him in the use of some other software system. Our interest is centered on the problem of generating the user interface software with the assistance of an interface engineering environment.

Perhaps that the single most important criterion by which a user interface system should be judged is the degree to which it facilitates the accomplishment of a particular task by a user with little or no previous experience. For that reason we require our interface engineering environment to be knowledge based. In fact we require it to store and handle knowledge about the application domain, about its users and even about the task of designing user interfaces. We have developed such an environment, which we have named PUC, the acronym of our university. It also stands for Planned User Communication.

PUC's requirements are based on our view that the ideal environment for designing a personalised interface is one which involves the collaboration of the user and an interface designer

who knows about the application. Although not yet completely solved the problem of designing a tailor made interface software for a custom application is, by far, a simpler problem. In fact, some start the software life cycle process by specifying the interface [1] as an approach to software prototyping in general. Our assumption, in turn, is more complex because we are addressing the issue of tailoring interfaces to general software packages so as to make them adjusted to individual users profiles and preferences. The approach used in the PUC project is to "incorporate the interface designer to the environment" and have the user produce his/her own interface with the assistance of the environment. The approach (which will be subject to extensive experimental evaluation) aims to allow the end user to plan his/her future interaction with an application while learning about it.

In figure 1 we schematically introduce the main functional blocks that constitute the environment and its output. The interface designer is represented inside PUC through a model of the particular application (AM) for which the interface is to be generated and a model formed by rules about interface design principles (IDM). The former is introduced into the system by an expert on the application. The latter contains some established rules about interface design (as in [7]) and interacts with the part of the user model that contains information about the particular user's interaction style (UM/IS).

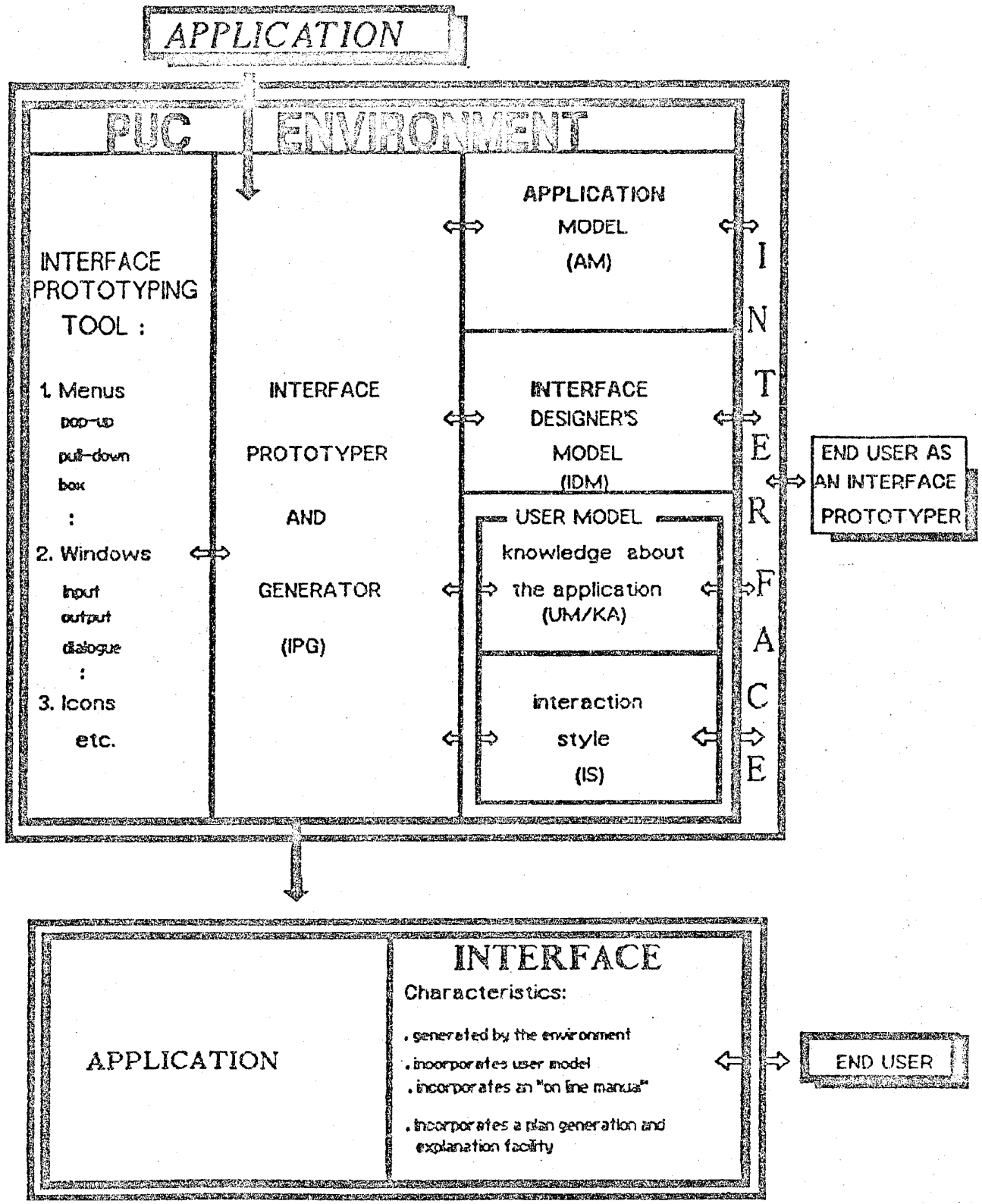

figure 1
Overview of PUC

The user modelling capability of PUC covers two aspects: user knowledge about the application (UM/KA) and user interaction style (UM/IS). The knowledge acquisition system (UM/KA) captures the user's knowledge about the application while he/she interacts with PUC (if the end user is not a novice). The model is to be completed when the user interacts with the application that carries the final interface (see bottom part of the figure). The user interaction style (UM/IS) is modelled as the environment learns about how the end user specifies the interface using the interface prototyping tools (IPT).

The knowledge bases (AM,IDM,UM/KA and UM/IS) interact with the interface prototyper and generator (IPT). This part of the system allows the user to specify the layout of the interaction screens that he/she wants to use in connection with the application. The IPG has access to several tool bases that include collections of standard routines to handle menus, windows, icons etc. So far the project has produced two preliminary implementations ([2], [3]) of the generator. The key questions being examined are how to structure the application software for use in connection with the environment and which notation should be used for specification of the interaction. In the present paper we will concentrate on the knowledge based aspects of the environment.

In the following sections we describe the end user interaction with the knowledge bases that form part of the environment, and give some details about how knowledge is

represented in each model. At the end we summarize the current status of the environment architecture.

2. Representation of the Application Model

For the end user to be able to prototype his/her own interface using PUC an expert in the particular application considered has to provide the system with an application model.

We use generic predicates to describe the application. The required predicates, together with their description are presented as follows:

. is-composed-of (Concept,Concept)

defines concepts which are combinations of other concepts;

. concept (Concept)

defines all concepts used in the application;

. function (Function)

defines all functions used in the application;

. statement (Command,Function,Concept)

specifies which commands or set of keys on the keyboard execute **Function** based on **Concept**;

. explain (Function,Concept,Explanation)

explains how and/or when to use **Function** based on **Concept**;

. macro (Function,List-of-functions)

allows the designer to group a set of functions.

In most applications, many interaction statements require a pre-condition. To deal with this problem and provide a way to represent a given sequence of commands, the following predicate is used:

change-of-state (Function1,Concept1,Function2,Concept2)

specifies a pre-condition to a particular application; Function1 can be instantiated to a particular value named "provide" which means that an interaction with the end user will be required at this point during interaction.

To arrive later at the user model, the concepts and the functions in the application model are organized in a hierarchical way by the designer (as in [6]). The predicates that describe this structure are:

. hierarchy-concepts (concept-tree)

. hierarchy-functions (function-tree).

We also include a predicate for capturing information about which users know about which interaction statements in the application, that is, for defining stereotypes ([4],[5]).

The format of this is:

. model (User-type,Command,Function,Concept)

Most of what follows will be based on the representation that results from the instantiation of the above predicates. The example at the end helps to clarify the use of the model.

3. End User Interaction with the Knowledge Bases: The Built-in Interface Designer

As described in the introduction, the central idea of PUC is to have a model of the interface designer as part of the environment assisting the end user on the elaboration of a personalized interface to a given application . The purpose of the simulated interface designer is to accompany and guide the user during the specification of the interface. There are three types of action taken by the built-in designer:

- for a novice user it will suggest an interface based on the application model ,
- while interacting with more advanced users it provides guidance on the design of the interface,
- when the interface has been specified it checks its consistency according to the rules of the application model and fills possible gaps to produce the final interface.

Before we comment on the end user interaction with the environment we must recall that the environment will classify the end user according to his/her knowledge of the application. We use three levels of classification : novice, intermediate and expert.

The process of interaction between the end user and PUC starts with the question of whether the user knows the application. From this point on, the procedure below is followed.

IF user does not know the application THEN
- classify the user as novice,
- invite the user to take part in the interface specification
- IF user is not participating THEN
  - produce the interface based on the application model,

  ELSE (user is involved)
  - produce interface interactively with the user (executing simulations of the prototype interface),

ELSE (user knows the application)
- follow the interface specification being proposed by the user while interfering with suggestions; the user may require help such as:
  - suggestion of screen layouts,
  - consistency analysis of all or part of the interface under development,
  - simulation

From now on we are going to describe how the human designer is simulated in the three situations indicated in the central control procedure and the case where it generates the final interface (after consistency checking).

The automatic generation of the interface is illustrated at the end through the example presented in section 5.

The second situation described in the control procedure deals with a novice user that interacts with the interface prototyping and generation. In this case while the interface is being automatically generated the user can interfere by proposing changes. The user proposals made during this phase will become part of the knowledge base about the interaction style (IS). It will be taken into consideration during continuation of the interaction.

The third case in the control procedure, considers that the user will produce its own interface. Here the built-in designer of

PUC will also be involved, albeit less visibly, with the interface design. The following rules and actions illustrate the designer involvement with the interface specification (they use resources already implemented in [2] and [3]).

IF menu system is used THEN
- analyse the list of items that will form the menu fields: the list should be homogeneous,
- guarantee that no repetitions occur,
- suggest the selection technique (capital letter, highlight etc) and record the preferred approach,
- check the position and presentation format of the names of menus fields that form a tree of menus (ex.: pop-up linked with pull-down linked with etc),
- validate the sequence of menus according to the predicate "change-of-state".

IF window system is used THEN
- analyse window space versus content: if window is loaded suggest sub-divisions based on existing knowledge about the application, if window is empty suggest reduction.

IF function line is used THEN
- list should be homogeneous as in the menu case,
- if there exists more than a certain number of fields, suggest a last field named "others" or "more" linked with another function line,
- if a selection involves more than one key on the keyboard suggest other alternatives.

When the user is finished with the specification another set of rules is applied to detect the gaps in the specification with respect to the application model. It works by detecting all pairs function-concept that the application is capable of executing and that were not included in the interface design. To each pair function-concept the following needs to be done:

- if **Concept** appears in the hierarchical tree of concepts among those which have been used in association with

Function it must also be considered in the specification; it triggers the process of inclusion of this new item in a menu;

- a similar process takes place in the case of **Function**,

- if the pair exists in some predicate "change-of-state" try to position it in the appropriate point in the sequence of screens; if this new function has as its pre-condition the particular function valued "provide" a dialogue window should be linked to the function,

- if there is a group of unspecified functions or concepts refer to the interaction style to include the new group according to these standards.

4. The User Model

The concept of user model is dealt with in the PUC environment from three diferent points of view. We talk about modelling the end user while he/she display some knowledge about the application and/while he/she uses the interface construction tools during the process of interface specification. These first two aspects of the user modelling process are used by the built-in designer to interact with the user as described in the previous section. The third user model that is incorporated into the interface produced by the environment is meant to customize the process of planning with explanation generation avaliable to the user when he/she interacts with the final interface (see figure 1)

In the present section we will deal first with user modelling in PUC from the first point of view: how much he/she knows about the application.

Whenever a user refers to an application function during the interaction with the environment some information can be drawn about how much he/she knows the application. The system generates predicates called "individual-model" based on the information. The predicates have the form:

individual-model (Command,Function,Concept).

The number of the above predicates will depend on the user's level of expertise. The model as described is transfered to the final interface to the application. During that phase the user modelling process proceeds in the following way.

```
IF the user applies a pair function-concept THEN
     .  store the information in the corresponding
        "individual-model" predicate,
     IF the pair function-concept belongs to the predicate
        "model" and the parameter "User-type" is higher than
         current value THEN
         . store a new predicate entitled
                 future-user-type (User-type)
           which stands for the higher classification level,
         .  IF the number of predicates referring to the
           level above the current level is fifty percent
           (or more) of the functions in the AM THEN
              .  update the predicate "user-type" that sets
                 the level of the user's expertise.
```

The utilization of the user model in conection with the final interface allows the generation of plans adjusted to the users expertise. Using the predicates generated above, that is

individual-model (Command,Function,Concept),

user-type (User-type)

and the predicates in the application model

change-of-state (Function1,Concept1,Function2,Concept2)

command(Command,Function,Concept)

explain (Function,Concept,Explanation)

the plans are generated according to the procedure below.

WHILE there is a pre-condition to the goal pair function-concept, that is, Function1 and Concept1 DO
- find its pre-condition, that is, Function2 and Concept2, in the predicate "change-of-state" and display it,
- IF the user does not know the pre-condition THEN
  - find the corresponding command and display it
  - IF he/she is a novice find the explanation and show it,
- update the goal pair, that is, Function1 and Concept1 with the corresponding values of Function2 and Concept2.

5. A Simple Example

Any sequence of interactions with an environment with features such as the ones incorporated in PUC is hard to summarize within the length of a technical paper. For that reason we have decided to concentrate here on the application model (section 2) and the built-in designer's capability to directly generate a given interface (thereby avoiding representing the other situations which involve too many interactions with the user).

The application chosen is a WordStar-like text editor. We start by showing a sub-set of the application model that the human designer needs to provide to the environment. The particular instantiation of the predicates presented in section 2 is as follows:

```
concept (character)
concept (word)
concept (line)
concept (block)
concept (page)
concept (file)

is-composed-of (file,block)
is-composed-of (file,page)
is-composed-of (page,block)
is-composed-of (page,line)
is-composed-of (block,line)
is-composed-of (line,word)
is-composed-of (word,character)

function (remove)
function (move-forward)
function (move-backward)

macro (every-other,[remove,move-forward])

command (F3,every-other,word)
command (<--,move-backward,character)
command (-->,move-forward,character)
command (CTRF F,move-forward,word)
command (CTRL A,move-backward,word)
command (PGUP,move-backward,page)
command (PGDN,move-forward,page)
command (CTRL Y,remove,line)
command (DEL,remove,character)
command (CTRL T,remove,word)

change-of-state (search,word,provide,word)
change-of-state (provide,file,edit,file)
change-of-state (edit,file,save,file)
change-of-state (edit,file,move-forward,word)
change-of-state (edit,file,move-backward,word)
change-of-state (edit,file,move-backward,character)
change-of-state (edit,file,move-forward,character)
change-of-state (edit,file,remove,word)
change-of-state (edit,file,remove,line)
change-of-state (edit,file,search,word)
change-of-state (provide,environment,enter,environment)
change-of-state (enter,environment,edit,file)
change-of-state (enter,environment,exit,environment)
change-of-state (enter,environment,print,file)

explain (remove,word,"remove all character ...")

etc.
```

The human designer defines the user stereotypes with respect to the application by using predicates such as:

```
model (novice,<--,move-backward,character)
model (novice,-->,move-forward,character)
model (novice,PgUp,move-backward,page)
model (novice,PgDn,move-forward,page)
model (novice,DEL,remove,character)
model (intermediate,CTRL D,move-forward,character)
model (intermediate,CTRL E,move-backward,character)
model (expert,F3,every-other,word)
```

The designer is also responsible for the definition of the hierarchical structure of concepts and functions:

```
hierarchy-concepts ([file,[block,[word,[page,
                      [line,[character]]]]]])
hierarchy-functions ([every-other,[remove,[move-backward,
                       move-forward]]])
```

From this point on the environment interacts with the end-user to determine the preferred interface characteristics. In the present simple example we will illustrate the situation in which the interface is generated automatically while following suggestions from the user. In this case:

- the system searches for the most general concept (it reaches "file" through the predicate "hierarchy-concept"),

- the system finds that the concept "file" in the "change-of-state" predicate has as its pre-condition the concept "environment" which, in turn, is associated with the particular function valued "provide".

- the environment, at this point, prompts the user:

  Environment ?

  The user provides the name of the application: EDITOR.

- the system then groups all functions that have as a pre-condition the condition of the pair "provide,environment"

in the "change-of-states" predicates. In the example it applies to the following pairs:

```
edit,file
edit,environment
print,file.
```

It produce the pop-up menu in figure 2.

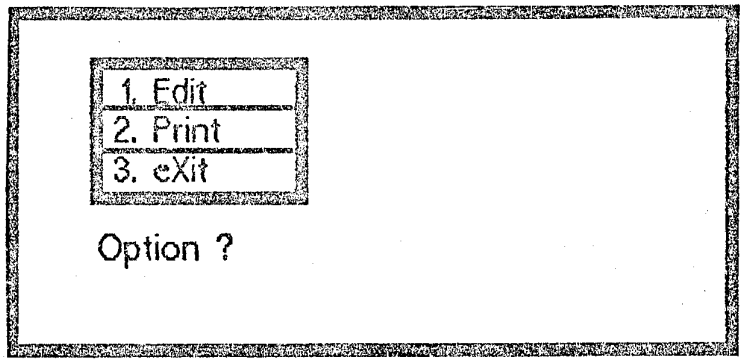

figure 2

The user's options may be to select the number or the capital letter directly. He/she can also prefer to use the highlight technique.

the system will decide next which interface tool (see figure 1) it is going to use in association with each of the fields of the menu in figure 2.

Both "edit" and "print" (menu in figure 1) have as a pre-condition the particular function valued "provide" in a "change-of-state" predicate. It will lead to a dialogue window of the type:

The "exit" field (figure 2) has no pre-condition. It is a direct execution command and so its selection means its execution.

The "edit" field besides having a pre-condition, is itself a pre-condition for all of the application functions. The previous dialogue window will link to screen in figure 3.

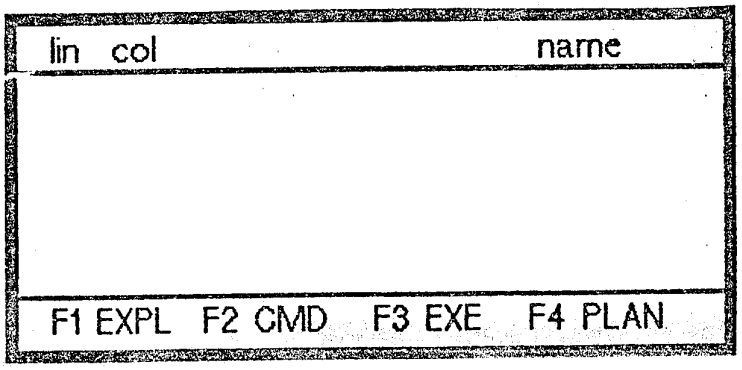

figure 3

The selection of any option in the function line above (figure 3) will trigger the use of pull-down menus in the upper part of the window shown in figure 4.

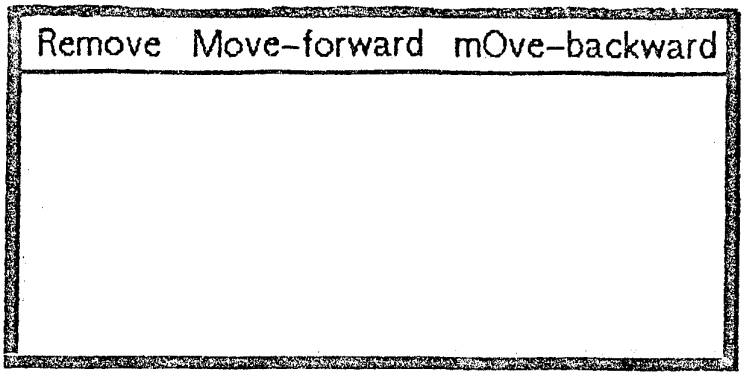

figure 4

The menu names in figure 4 are the functions specified by the human designer. For each name there is a menu containing the concepts related to the functions.

The interface generation would go on in a similar way by using operations such as the ones illustrated above (which are already included in the experimental implementation [2]).

6. Conclusions

We have described here many of the knowledge-based features of the PUC environments. Emphasis was placed on the central feature of the system: the transference of knowledge from an expert interface designer to an end user via the knowledge bases that compose the environment.

The preliminary models of the environment's knowledge bases have been designed together with the procedures to incorporate this knowledge, to the process of producing an interface. Interface generators were developed independently and are now been used to study the behavior of interface designers while interacting with users in the process of prototyping an interface.

The final implementation of the complete PUC environment will be instrumented with various probes to assist in its evaluation by a wide population of users.

REFERENCES

[1] University of Hawaii - "User Interface Prototyping Tool for Workstation-Based Systems",Informal Report, 10 th International Conference on Software Engineering, Singapore,1988.

[2] Lisandre,B.;Santos Filho,H.M. - Preliminary Report on the Experimental Implementation Beta, Technical Report of the PUC project, 1988.

[3] Duarte,R.C.;Levy,C.H.;Silva,N.S. - Preliminary Report on the Experimental Implementation Alpha, Technical Report on the PUC project, 1988.

[4] Rich,Elaine;"Stereotypes and User Modelling",preliminary report,july,1987.

[5] Rich,Elaine; "User Modelling via Stereotypes", Cognitive Science 3, 329-354;1979.

[6] Rich,Elaine; Users are Individuals: individualizing user models; Int.J. Man-Machine Studies, 1983.

[7] Shneiderman,B.;"Designing the User Interface: Strategies for Effective Human-Computer Interaction, Addison-Wesley Publishing Company, 1988.