Human–Computer Interaction – INTERACT '87
H.-J. Bullinger and B. Shackel (Editors)
Elsevier Science Publishers B.V. (North-Holland)
© IFIP, 1987

PAC, an Object Oriented Model for Dialog Design

Joëlle Coutaz
Laboratoire de Génie Informatique (University of Grenoble)
BP 68
38402 St-Martin-d'Hères Cedex, France
UUCP ...!mcvax!inria!imag!joelle

PAC is an implementation model that attempts to bridge the gap between the abstract sphere of theoretical models and the practical affairs of building user interfaces. It takes as a basis the vertical decomposition of human-computer interaction into semantic, syntactic and pragmatic layers as promoted by some theoretical models. However, PAC stresses the fact that these notions do not form strict monolithic layers but are distributed across related "chunks", called interactive objects. For doing so, PAC recursively structures an interactive application in three parts: the Presentation, the Abstraction and the Control. The Presentation defines the the concrete syntax of the application whereas the Abstraction corresponds to the semantics. The Control maintains the mapping and the consistency between the abstract entities and their presentation to the user. The Presentation of an application is in turn decomposed into a set of interactive objects, entities specialized in man-machine communication. As for applications, an interactive object is organized according to the PAC model. PAC has been used for the construction of two interactive applications and is currently applied to the development of a User Interface Management System.

1. Introduction

Models for the description of human-computer interaction come essentially in three forms: high level theories, engineering models and implementation models.

High level theories such as GOMS [1] and CLG [2] offer a way to understand the nature of human-computer interaction and help organize the design process. Engineering models such as the Keystroke Level Model [3] and Phyllis Reisner's work [4] are useful in making predictions about a particular design or evaluating alternatives. Implementation models provide application designers with a practical framework for building user interfaces.

If one considers the implementer's point of view, theoretical models are too abstract, engineering models come too late and implementation models are too functional. Theoretical models do not help much in the practical needs of the implementer who is not necessarily a knowledgeable person in the field of human-computer interaction. Engineering models are useful for the refinement of user interfaces not for their initial definition. Finally, implementation models organize functional pieces of software without integrating the "substantial marrow" of theoretical models.

This paper presents PAC, an implementation model that attempts to bridge the gap between the abstract sphere of theoretical models and the practical affairs of building "user-integrated" user interfaces. The next section introduces the abstract model that served as a basis for PAC. Section 3 and 4 describe PAC itself an show how the model provides the foundation for supporting some of the elements essential to the quality of a user interface: context, concurrency and adaptability. Section 5 illustrates the use of PAC for the construction of two interactive applications and section 6 compares PAC to related work.

2. The Abstract Foundation of PAC

Abstract models all structure the human-computer interaction into stages [5, 6, 7]. Unfortunately, the nature of these levels varies in a subtle way depending on the cognitivist or linguistic background of the model designer. However, one can identify some commonalities between these perspectives, in particular the notions of semantics, syntax, lexeme and physics. The picture compiled by Norwood Sisson in [8], integrates nicely the different views by showing how each human cognitive stage has its corresponding level in the user interface software.

PAC takes as a basis both the vertical decomposition and the horizontal correspondance of Sisson's model. However PAC stresses the fact that the notions of semantics, syntax, and so on ... do not form strict monolithic layers. Instead, each notion is distributed into related "chunks", called objects, as is described in the following section.

3. PAC, an Implementation model

PAC structures an interactive application in three parts: Presentation, Abstraction and Control.

- the Presentation defines the concrete syntax of the application, i.e. the input and output behavior of the application as perceived by the user;
- the Abstraction part corresponds to the semantics of the application. It implements the functions that the application is able to perform. These functions are supposed to result from a thorough task analysis;
- the Control part maintains the mapping and the consistency between the abstract entities

(involved in the interaction and implemented in the Abstract part) and their presentation to the user. It embodies the boundary between semantics and syntax. It is intended to hold the context of the overall interaction between the user and the application.

This constitutes a first level of description of an interactive application. We need now to refine the Presentation part.

The Presentation of an application is a set of entities, called interactive objects, specialized in man-machine communication. As for applications, each interactive object is organized according to the PAC model. Consider for example the pie chart shown in figure 1:

1. the Presentation is comprised of:

   • for output, a circular shape and a color for each piece of the pie,

   • for input, mouse actions that the user can perform to interactively change the relative size of the pieces;

2. the Abstraction is comprised of an integer value within the range of two integer limits;

3. the Control maintains the consistency between its Presentation and Abstraction. For example, if the user modifies the size of one piece, the Control part provokes the update of the integer value. Conversely, if the application or another interactive object modifies the value of the integer, the size of the pieces is automatically adjusted.

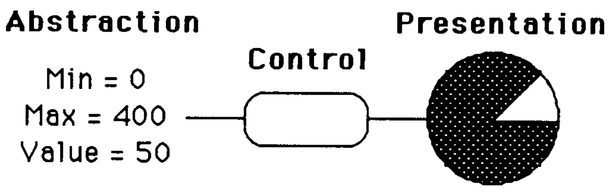


Figure 1: An elementary Interactive Object

Compound objects can be built from elementary interactive objects and structured as PAC entities. The Abstract part of a compound object reflects its semantics; the Presentation is partly determined by the presentation of the components and partly defined by specific attributes; the Control serves for all-purpose local maintenance. Consider, for example in the figure 2, the super-pie-chart made of the elementary pie chart described above and of a numeric string. If the control, say C, of the super-pie-chart receives a message from control C1 of the elementary pie chart and if this message notifies C of the modification of the abstract value of the elementary pie chart, then the role of C is to tell C2, the control of the numeric string, to update its value. This is a very simple example of how composition of objects can lead to arbitrarily sophisticated objects.

By applying PAC recursively at every level of abstraction of the user interface, everything in an interactive application is a PAC object: from the elementary interactive object to the whole application. As shown in the upper rectangle of figure 3, the whole interactive application is a PAC entity. The Abstraction part of the application involves three domain dependent concepts in the dialogue. The Controller at the top of the hierarchy bridges the gap between the Abstraction and the Presentation. The Presentation is made of 4 interactive objects. The second lower rectangle shows the PAC structure of the compound interactive object represented as a black circle. This object is built from two elementary PAC objects and one compound object which, in turn, is composed of two elementary PAC objects.

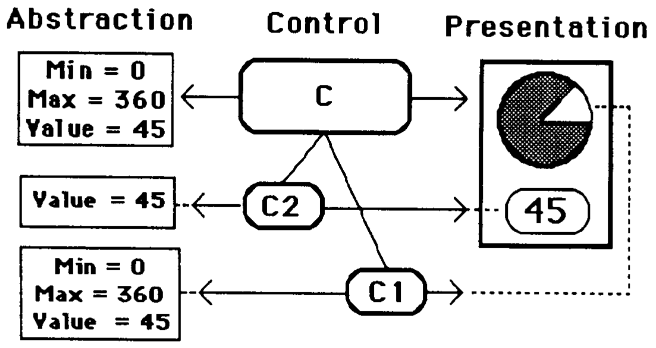


Figure 2: A Super-Pie-Chart is a compound object made of two elementary objects: a pie chart as described in the figure 1 and a string.

In addition, the user interface of a workstation (generally refered to as a shell) may be modelled in a straighforward manner by adding an extra PAC layer on top of the application level. The Abstract part of that layer may include such global data structures as the "clipping board" or the "network status". The Presentation would present these data structures and allow for the initial invocation of applications. Finally, the Control part would, of course, bridge the gap between the abstract and the concrete sides. It would as well supervise the control parts of all of the active applications. Such an arbitrator should provide the basis for a uniform mechanism for transferring data between applications.

This recursive object-oriented organization presents some advantages which are described in the following section.

4. The Interest of PAC

The interest of the PAC model is three-fold: a consistent framework, the notions of control and interactive object.

4.1. A Consistent Framework

PAC defines a framework for the construction of user interfaces that is applicable in a consistent way at any level of abstraction. As a result, semantics, syntax and "pragmatics" are distributed across a hierarchy of entities rather than forming hermetic layers that do not match the cognitive organization of human knowledge [9].

4.2. The Notion of Control

PAC distinguishes functional notions from presentation policies but introduces the notion of control to

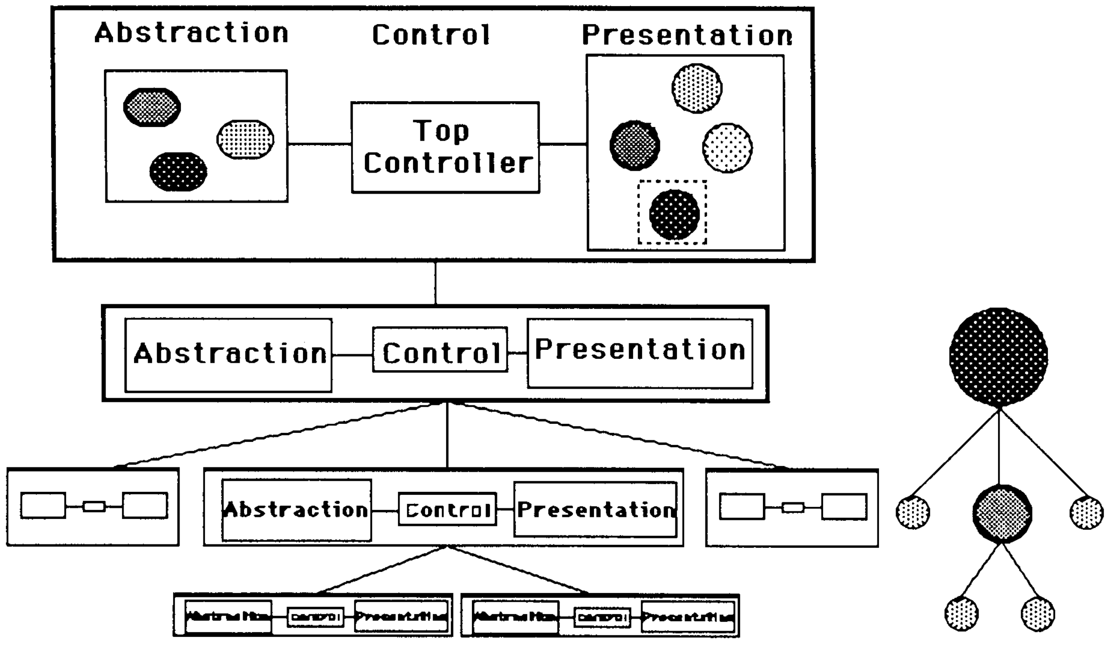


Figure 3: An interactive application structured according to the PAC Model.

bridge the gap between the two worlds. This link is important: even minor syntactic details require knowledge about the semantics. Therefore the role of the control part is to maintain an extensive knowledge about the two worlds it serves. As shown in figure 4, this knowledge can include contextual information that may be useful for history, help, error explanation, multi-thread dialogue and automatic adaptation to the user.

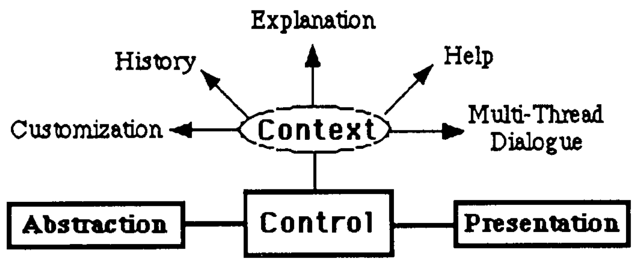


Figure 4: Contextual information that a Control part may hold.

To illustrate the role of the notion of Control, let us take two simple examples: a string object used as a field in some form, and the top controller which links the application to its user interface. The string controller may keep track of the values that the user has successively entered during the session. The user is then able to retrieve the various values of the field by issuing the "next" or "previous" commands (for example, by typing Ctrl-n or Ctrl-p, if an Emacs-key-translation-table is attached to the string) or even select the values through a pop up menu attached to the string.

For reasons justified in [10], the Top Controller should hold the main control loop of the dialogue but hooks should be provided for mixed control. Mixed control is required by applications when data are needed from the user during the processing of a function (e.g. the popular Unix "more" command). The problem is that the control must not stay in the application hands because the user should not be bound to directly satisfy the request. He should be able to invoke several functions of the application before replying to the initial request. For doing so, the Abstract application can issue READ messages whereas the Top Controller maintains dialogue state variables. On reception of a READ message, the Top Controller creates a request object and activates the interactive objects that represent the abstract entities referenced in the READ message. The request object, a dialogue state variable, watches the interaction between the user and the interactive objects and detects when the user has satisfied or cancelled the READ. When the READ operation is completed, control automatically returns to the function that issued the READ.

4.3. The Notion of Interactive Object

PAC takes full advantage of the object-oriented paradigm with the notion of interactive object in particular for customization, concurrency and distribution.

Interactive objects are customizable. Object-oriented programming languages support data abstraction which makes it possible to change underlying implementations without changing the calling programs. In our case of interest, this principle allows the internal modification of an interactive object without changing its presentation and abstract interfaces. Interestingly, it also allows the modification of one interface without any side-effect on the other one. For example, one can modify the presentation of an interactive object (such as attaching a different key translation table to an interactive object of type string) without reflecting on its abstract behaviour. This property makes it possible fine grained explicit adjustments of the user interface without questioning the presentation of the whole application. A second type of adjustment is the automatic adaptation to the user. The object paradigm seems suitable for the implementation of an intelligent observer where each interactive object would model the user with its own set of rules as made possible in languages such as Loops [11].

Interactive objects are active concurrent entities. They evolve, communicate and maintain relationships with each other. Such activity, parallelism and communication are automatically performed by the Object Machine, the generic class of the interactive objects. The Object Machine defines the general functioning that is made common to all of the interactive objects by means of the inheritance mechanism. In particular, each object owns a private finite state automaton for maintaining its current dialogue state. On receipt of a message, an object is thus able to determine which actions to undertake according to its current state.

Interactive objects implement the dialogue in a distributed way. This feature can serve as a basis for the implemention of facilities related to notions that appear to be everywhere such as every form of contextual information. The examples presented in 4.2 illustrate the kind of contextual information that can be held by interactive objects. Distribution also provides the necessary grounds for concurrent multiple I/O in the following way. The set of automata (one automaton per interactive object) defines the global state of the interaction between the user and the application. The control of the interaction is therefore distributed in an evolutive network of interactive objects. Dialogue control is not handled by a unique monolithic dialogue manager difficult to maintain, extend and implement, in particular when one wants a pure user-driven style of interaction. Conversely, since interactive objects are able to maintain their own state, it is easy to let the user switch between objects in any order. Thus, an object-oriented approach provides for free the maintenance of the user's arbitrary manipulations.

The notion of interactive objects and the modular dichotomy between domain dependent concepts and presentation policies have been successfully applied to the development of two interactive applications. These are briefly presented in the following section.

5. The Use of PAC

Our first experience with PAC is the implementation of an application framework that provides application programmers with a reusable and extensible skeleton [12]. This skeleton has been implemented in C on top of the Macintosh toolbox and has been re-used for two applications: a simple graphics editor and a graphics terminal emulator. It is currently used for the implementation of a tool for interactive dialogue specification.

According to the PAC model, the reusable skeleton is comprised of three parts:

1. the Abstract part which serves as the interface to the domain dependent functions. This interface is made of a predefined set of entry points that the application programmer fills in to access the semantic functions per se;

2. the Controller which contains the top level loop and acts essentially as a mediator between the underlying system, the Abstract, and the Presentation parts. It receives low level events generated by the system, dispatches them to the Presentation and, for events that have semantic side-effects, calls the appropriate entry point of the Abstract side.

3. the Presentation part which is comprised of a set of basic classes of objects, such as window, menu and serial line classes. These classes are enriched versions of the standard Macintosh classes, providing the application programmer with extended facilities such as automatic refreshing and scrolling of windows content. These objects process the low level events transmitted by the Controller as far as they can. If an object cannot fully process an event (for example, a mouse click in a window area that is not meaningful to the window class), it generates an abstract event that the Controller passes to the Abstract side for further processing (for example, the mouse event just mentioned generates an "INCONTENT" abstract event that is directed to the entry point "AbInContent" to which an application dependent function is linked).

Given this architecture, switching from one application to another requires implementing domain dependent functions and connecting them to the predefined entry points of the Abstract part. In our own experience, switching from the "MiniMacDraw" program to the emulator required adding a few new entry points. This extension turned out to be easy to perform, due to the rigor imposed by the underlying PAC model combined with the object oriented paradigm.

Figure 5 shows the screen produced by a robot simulator [13] running on a VAX and sending requests through a serial line to the graphics terminal emulator running on a Macintosh. The top left window is a classic VT100 emulator with automatic vertical and horizontal scrolling. The other windows show a graphical representation of the robot's knowledge about its environment. Here, the PAC skeleton handles the serial driver for the graphics terminal emulator, takes care of the syntactic tasks by automatically refreshing the windows when they are resized, scrolled or moved around by the end-user, signals invocation of semantic functions (such as a baud rate change) and adheres, for the application, to the standard Macintosh recommandations such as the access to desk accessories and the "About" window. The next step of this experience is to encapsulate the functionalities of the terminal emulator into a new class of presentation object.

The application of the object-oriented paradigm to the construction of user interfaces is rather new but not specific to PAC. The following paragraph presents similar works.

6. Related work

Other models than PAC rely on the notion of object and make the distinction between functions and presentation. These are the Smalltalk MVC model [14], Ciccarelli's PPS model [15], EZWin [16] and GWUIMS [17].

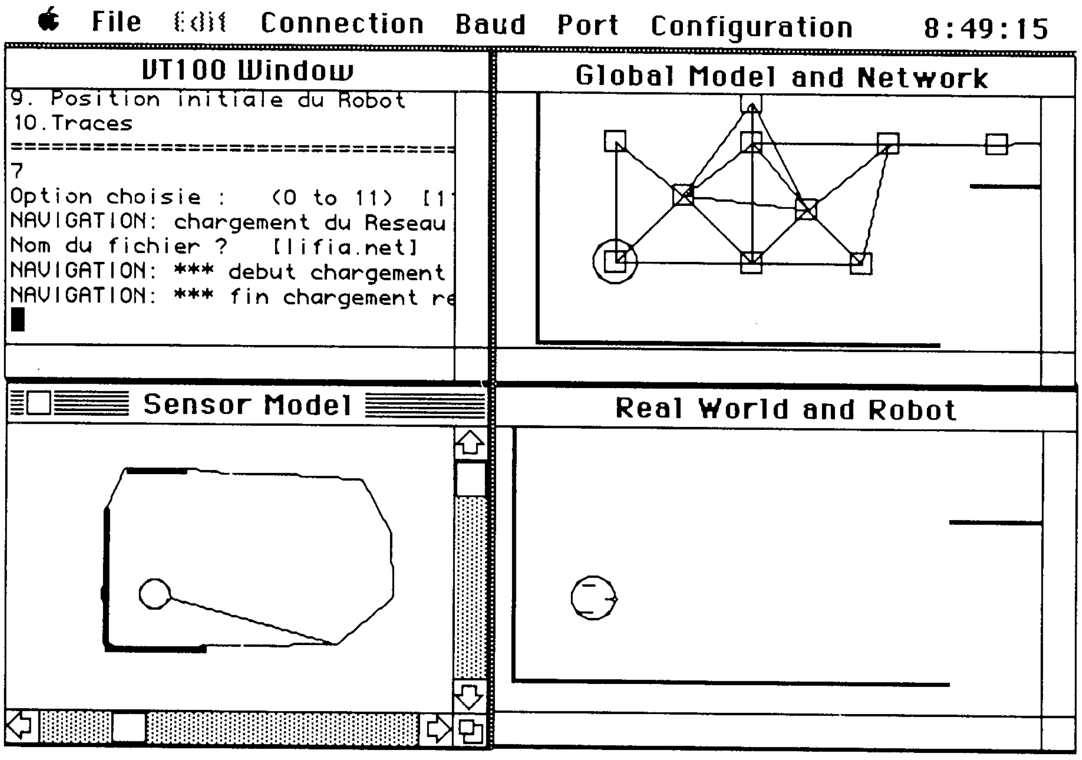
Model, View and Controller are three Smalltalk classes. A Model implements some functions, a View is the output rendering of a Model and a Controller is the input driver of a Model. PAC and MVC differ in two points:

• PAC distinguishes but encapsulates functions and presentation into a single object. A local controller encompasses the boundary between local semantics and local syntax. At the opposite, MVC makes an explicit use of three Smalltalk objects which must maintain their consistency through message passing. In MVC, the notion of control is diluted across three related objects whereas it is explicitly centralized in PAC.

• PAC combines input and output behaviour into one component whereas MVC distributes them across two objects. The distribution has the advantage of flexibility (one can change the input syntax without disturbing the component dealing with the output syntax). Unfortunately, it is often the case that, at the fine grained level of the interaction, input events are strongly related to immediate output feedbacks.

part can maintain a context (e.g. current status, validity, defaults) that can be usefully exploited during interaction to provide the end user with dynamically tuned informative feedback.

GWUIMS revolves around five types of objects: A-objects embody the semantics of the application; I-Objects bridge the gap between A-Objects and R-objects. R-objects control the presentation. They are at the boundary between the lexical and syntactic levels of a UIMS. Lexical input is carried out by T-objects whereas lexical output is performed by G-objects. An elementary PAC object is functionally equivalent to the combination of a R-Object with a T-Object and a G-Object, and the MOUSE Controller is functionally equivalent to I-Objects.Clearly, PAC and GWUIMS are closely related in their overall organization although GWUIMS splits the presentation into input and output objetcs.



```
9. Position initiale du Robot
10.Traces
================================
7
Option choisie :   (0 to 11)  [1
NAVIGATION: chargement du Reseau
Nom du fichier ?   [lifia.net]
NAVIGATION: *** debut chargement
NAVIGATION: *** fin chargement re
```

Figure 5: A View of the Robot Simulator whose architecture is based on PAC.

PPS is organized around a network of Presenters and Recognizers that manipulate two types of data bases. An Application data base defines some functions and a Presentation data base presents the Application data base. A Presenter builds a Presentation data base from an Application data base. The user modifies a Presentation data base, the associated Recognizer interprets the user's actions and modifies the corresponding Application data base. As in MVC, PPS makes a dichotomy between input and output behaviour but, at the opposite of MVC, these behaviours are not carried out by objects. PPS objects are passive entities manipulated by Presenters and Recognizers.

EZWin objects are close to PAC interactive objects: both combine input and output policies. The originality of PAC objects is the existence of the Control part which lays the ground for the implementation of distributed notions related to the notion of control. In particular, the Control

7. Conclusion

PAC is currently applied to the development of a User Interface Management System [12] implemented in C on the Macintosh. It is an attempt to integrate in a natural way, concepts provided by abstract models such as the notions of semantics, syntax and pragmatics, and practical guidelines derived from experiments such as "let the user drive and customize freely the dialogue". Now, the next step in the refinement of PAC is to provide the implementer with more details about the nature and the arrangement of the contextual information hold in the hierarchy of controllers; in particular identify which contextual information are common to all systems and which ones are domain specific.

Acknowlegment. This work is part of the project GUIDE, an object oriented distributed system, which is being designed at the Laboratoire de Génie Informatique (IMAG, University of Grenoble). It sets the foundation of the user interface for the GUIDE workstations.

References

[1] S. Card, T. Moran, A. Newell: The Psychology of Human-Computer Interaction; ISBN 0-89859-243-7, Lawrence Erlbaum Associates, Publish., 1983.
[2] T. Moran: The Command Language Grammar: a representation for the user interface of interactive computer systems; International Journal of Man-Machine Studies, 15, 1981, 3-50.
[3] S.K. Card, T.P. Moran, A. Newell: The Keystroke-Level Model for User Performance Time with Interactive Systems; Communications of the ACM, 23(7), July 1980, 396-410.
[4] P. Reisner: Further developments toward using formal grammar as a design tool; Proceedings of Human Factors in Computer Systems, Gaithersburg, Maryland, March, 1982, 304-321.
[5] J. Foley, A. Van Dam: Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Co., Reading, MA, 1982.
[6] D. Norman: Stages and Levels in Human-Machine Interaction; International of Man-Machine Studies, 21, 1984, 365-375.
[7] B. Schneiderman: Designing User Interface: Strategies for Effective Human-Computer Interaction; Addison-Wesley Publishing Co., Reading, MA, 1987.
[8] N. Sisson: Dialogue Management Reference Model; ACM SIGCHI bulletin, 18(2), October, 1986, 34-35.
[9] J.R. Anderson: The Architecture of Cognition, Harvard University Press, 1983.
[10] J. Coutaz: The Construction of User Interfaces and the Object Paradigm; to appear in the proceedings of the European Conference On Object Programming (ECOOP'87), Paris, june, 1987.
[11] D.G. Bobrow, M. Stefik: The Loops Manual; Tech. report KB-VLSI-81-13, Knowledge Systems Area, Xerox, Palo Alto Research Center, 1981.
[12] J. Coutaz : La Construction d'Interfaces Homme-Machine, Rapport de Recherche IMAG-LGI, RR 635-I, novembre 1986.
[13] J. Crowley, Navigation for an Intelligent Mobile Robot, IEEE Journal of Robotics and Automation, Vol 1(1), March 1985, 31-41.
[14] A. Goldberg, D. Robson: Smalltalk-80: The Interactive Programming Environment; Addison-Wesley Publ., 1984.
[15] E.C. Ciccarelli: Presentation Based User Interfaces, Technical Report 794, Artificial Intelligence Laboratory, Massachusetts Intelligence Laboratory, August, 1984.
[16] H. Lieberman: There's More to Menu Systems than Meets the Screen; SIGGRAPH'85, 19(3), 181-189.
[17] J.L. Sibert, W.D. Hurley, T.W. Bleser; An Object Oriented User Interface Management System; SIGGRAPH'86, 20(4), 1986, 259-268.