Building graphic user interfaces for Computer Algebra Systems.

Norbert Kajler

INRIA, Centre de Sophia-Antipolis
2004 route des Lucioles, 06565 Valbonne Cedex, France

Published in LNCS 429 Proc. of DISCO'90, pages 235-244, Capri, Italy, April 1990. Springer-Verlag.

1 Introduction

A quality user interface is nowadays a crucial element for a Computer Algebra System (in the sequel : CAS). First, it makes the use of main functionalities much easier for students and casual users. Second, it gives access to the whole range of possibilities of the system and makes programming easier for expert users. In all cases, it highly improves the visual comfort and offers many advantages during formulae editing. The aim of this paper is the concise study of the need for a CAS user interface. Furthermore, it represents an important and prior step to any effective realization.

2 Description of the existing user interfaces

According to their user interface, CAS can be classified in two large groups. The first one works on traditional video display terminals, inputs are typed on the keyboard and displayed in one-dimensional form, outputs are displayed in a pseudo two-dimensional form using alphanumerical characters combinations. Macsyma , Maple , Reduce , and Scratchpad II systems support this kind of interface. The second one needs a high definition graphical terminal with a pointing device (mouse), the display of both inputs and outputs are in two dimensions, formulae are drawn with accuracy, and the mouse can be used to select subexpressions from previous inputs or ouputs and insert them in the current edited formula. Few systems already include such an interface; however we can think of : GI/S [22], Mathematica [21] and MathScribe [17, 20]. Of the above, GI/S and MathScribe are the one which have the most sophisticated user interface. More precisely, they are both user interfaces built upon traditional CAS, namely Macsyma and Reduce. In order to have a reference for the sequel, we will begin by studying the main characteristics of MathScribe.

MathScribe is a graphical man-machine interface for the Reduce system. It enables the user to open many kinds of windows by pressing buttons on a control panel driving the whole interface. In each window, mathematical formulae appears in a specialized editor which includes the following possibilities and features : real time two dimensional edition, cut and paste, scrollbars, undoing, substitutions, local and global abbreviations, structured edition by menu. MathScribe also enables one to visualize the Reduce flags in a separate window, to generate some EQN or TEX code from a formula, and to plot two or three dimensions graphs. MathScribe requires an extensive use of the mouse to select syntactical structures through menus. Nevertheless, it is possible to type alternative keyboard forms during edition. The error messages from the interface are presented in a terminal emulator which can also be used as a traditional textual interface with Reduce. The setting of interface parameters is limited to static modification of some elements of the visual aspect of the interface (colors, fonts, and dimensions of the windows). Moving or enlarging a window requires the use the window manager. Buttons for iconification and lowering are provided for each window.

3 Problematic of the man-machine interface

GI/S and MathScribe are the first examples of a sophisticated graphical user interface for a CAS. They eases the Macsyma or Reduce systems access and bring a great deal of visual comfort.

However, many heterogeneous requests about user interface are brought up by different kinds of CAS users. An improved user interface would increase the use of CAS to various areas like education, engineering, and mathematical research. Consequently, the next step is to develop specific interface for each specific domain. At the same time, new software engineering techniques have been developed, including complex tools like graphical syntax-directed editors, and toolboxes for man-machine interface design. That is why a new methodology including the use of up-to-date software tools, and the generation of some important parts of the interface is now an interesting alternative to directly coding the interface in C for a particular CAS (as it is the case with MathScribe).

3.1 Existing tools

Building a man-machine interface for a CAS raises two major difficulties. The first one lies in editing formula in two dimensions. The second one deals with consistently combining a bunch of basic components such as editors, menus, graph plotters and so on. Related to these problems, more and more tools, including programming environment generation, are available and their development becomes one of the major challenges for the beginning of the next century. Some of these tools will be presented, including examples of available software and research projects under development.

Nowadays there are sophisticated editors of formulae such as Grif or The Publisher, as well as generators of graphical language based syntax-directed editors which can adapt to the handling of mathematical expressions. Gigas [5] is a prototype of it. It was built above the Cornell Synthesizer Generator [19]. It enables the generation of structured graphical editors by compiling an attributed grammar written in the Synthesizer Generator formalism. The drawing is the result of an incremental calculus on the semantical attributes included in the grammar. The whole works in an X Window System environment. Similarly, a system such as Centaur [11], designed to the generate programming environments, should in the future include such a tool. Among the commercialized systems we can also think of Graspin [1].

Most of the existing interfaces are directly built above toolboxes. A toolbox (also called toolkit) is a collection of high level functions based above the lowest level routines of a windowing system. There now exists a very large number of toolboxes. Some of them are proprietary (for instance the toolbox of the MacIntosh , that of the NeXT Machine, or those of the Lisp machines by Symbolics). Some others are linked to X Window System (Athena Toolkit [18], OSF/Motif [15], InterView [13], CLUE[1] [12], etc). The last ones are independent of any windowing system and may be adapted to several computer environments (Workstations, MacIntosh, IBM PC, etc). An example is Aïda [9], built on the LeLisp language. These toolkits are also differentiated by their programming language (C, C++, Objective C, Smalltalk or Lisp), their general conception (use of object-oriented programming for instance), and their content (some of them offer many variations to the basis graphical objects, some others include specialized display machines such as tree or vector editors).

Beyond the toolboxes, another kind of tool appears [14]. It is frequently named UIDS (User Interface Design System). The prototype is SOS Interface [8] (written in Lisp on MacIntosh.). For a short time a tool of this kind has been commercialized by the Ilog firm (which markets LeLisp and Aïda) : Masaï [10]. It is a graphical user interface editor which generates Aïda code. Masaï itself includes a user-friendly graphical user interface which enables one to realize interfaces rapidly and interactively. Moreover, the code generated may be hand-altered by editing the corresponding Aïda program or edited again under Masaï. Of course, that greatly improves the ability to modify or adapt such a generated interface. Examples of UIDS also exist for the Lisp Machine. In the X Window System domain many

[1] CLUE (Common Lisp User Interface Environment) is an object-oriented programming system suitable for developing user interfaces in Common LISP under X Window System. It is built above CLOS (Common Lisp Object System) and CLX (Common Lisp programmer's interface to X).

tools are under development (such as Egerie [2]). There is also an equivalent tool already available for the NeXT machine : Interface Builder.

A more general approach is proposed by the man-machine dialogue specialists through the concept of UIMS (User Interface Management System) [4, 16]. This time the point is to divide the user interface under the form of three modules made up in layers which strongly separates the application from its interface. The communication between one layer and another is done in an asynchronous way by function calls. The three modules have specific roles. The first one is the "application interface". Its role is to ensure the communications between the application and the other parts of the interface. In a system such as Serpent [3], it consists of a shared data base of the variables of the application usable by the user interface. The second one is the "Dialogue Control". It is responsible for all interactions which need no semantical control (such as the syntactical checking of the data given by the user) and imposes the behavior of the interface (the "feel"). The third module is the "Presentation Manager", it carries out the display operations and manages the visual aspect of the interface (the "look"). Corresponding to the UIMS concept we find development environments integrating specification languages which allow one to generate interfaces on this model. Such complete and operational tools are rare; however many studies are being held. As an example we shall mention the Alberta UIMS [7], Serpent [3] and UIMX.

3.2 Methodology

The techniques of man-machine interface have greatly progressed from the conception of MathScribe. That is why a new approach seems desirable to create user interfaces meeting the constraints of portability and upgrading capability.

Indeed, instead of designing the user interface under the form of a graphical extension of a given CAS, we can begin with the users' needs to realize a precise specification of the desired interface. This is advantageous as it yields an interface largely independent of a particular system. The second step should consist of building an interface meeting these specifications for a target CAS. It is carried out by bringing into play the best suited software tools available. These tools must be chosen according to the needs expressed in the specifications. Moreover they evolve very quickly; hence separating the interface from the CAS allows one to derive benefits from the progresses realized by these tools.

The advantages brought by the use of generating tools are well known : much higher productivity during the interface developing, reliability of the code, and the possibility to quickly and easily modify the code by altering the high level language used by the generating tool instead of the code itself. In return we must mention the risks of inefficiency and memory space wastage when the chosen tools are not optimized. However the positive evolution of workstations (both in terms of powers and memory space availability) and the coming of industrial versions of these software tools enable us to foresee the success of this approach. Furthermore, we must note that most of these tools are themselves under development. But the needs related to CAS interfaces are complex enough to interest the designers of these tools and to realize their perfection.

Thus the development of a man-machine interface for a CAS sets a new problem. It consists first in clearly defining the needs. This is the aim of the remaining part of this paper. Then the point will be to study in details the different tools, participate in their perfection and fix the choices related to the effective realization of the interface.

4 Specification of the needs for the user interface

4.1 Elementary manipulation of formulae

Multiple window editing of 2D formulae display is provided both through the keyboard and mouse-based events. Editing is interactive and real time on the screen, while interaction with the CAS results from plain request by the user.

- **Formulae display**
  The point is to take advantage of the high resolution graphic screen to display formulae in

the most pleasant form for the user, without performance penalty or additional constraints. A quality display implies the use of different size character sets and accurate positioning of graphical components. To be efficient the display must be made in an incremental way (i.e. only modified parts of a formula must be refreshed on the screen). This is not a trivial problem as small changes made in a complex formula can bring about major revisions of the whole formula drawing.

- **Cursor scanning**
  The cursor is a graphical symbol pointing out, on the screen, the place of the formula which is likely to receive the next insertion (there is one cursor in each editing window). Unlike traditional editors, it does not perform scanning left to right and top to bottom , but follows the syntactical structure of the formula.

- **Editing with the mouse**
  The mouse is intended to act globally : it makes block manipulation easier. The following editing facilities must be done easily and accurately with the mouse : positioning the cursor at the mouse location, selecting a block, and other usual block manipulations like cut and paste.

- **Marked block**
  This is a concept local to the interface. For the user, the goal is to select, using the mouse, a sub-formula in a mathematical editing window. Once marked, the block can be easily seen (for instance, it may appear in a rectangular colored area) and is used as an implicit parameter for a large number of operations (erase, copy, print, etc).

- **Editing with the keyboard**
  The keyboard is the most suitable device for the manipulation of elementary lexical units. It is used for moving the cursor throughout the formula, and deleting or inserting textual elements. The highest number of sophisticated interface functionalities must also be usable from the keyboard. More, the bindings of the editing commands with the corresponding key sequences may be altered and displayed in a window.

- **"Undoing"**
  During the edition of a mathematical formula, it is handy to be able to go back step by step, in order to retrieve the formula as it was at each stage of the edition (as the undo command of a text editor does). This mechanism will have to work separately in each editing window, and is independent of the history, which deals with the requests and the results of the CAS.

- **Transmission of the formula to the CAS**
  It happens on an explicit command and concerns the expression edited in the active window. As edition is syntax directed, the expression won't be transmitted to the CAS unless it is syntactically correct. The feedback expression sent in answer by the CAS could be displayed either following the request (as it is for all traditional CAS), either in another window dedicated to the answers of the system (and laterally pained with the first one) or in the same location, replacing the request.

4.2 Manipulation of complex formulae

A CAS usually handles very large formulae (thousands of characters). In most cases, displaying them just as they are is useless. So, an essential capability of the interface will be to master the size of these complex formulae in order to make them readable and suitable.

- **Displaying of long formulae**
  The problem is to display expressions longer than the width of the editing window. According to the user preferences, the expressions will be either cut in accordance with caesura rules used by mathematicians (like MACSYMA does), either partially displayed, with the window linked to

a horizontal scrollbar (solution adopted in MathScribe), or printed on a single line, without scrollbars, and using the reduction mechanism.

- **Reductions and expansions**
  The point is to substitute to a block in a formula a graphical object which denotes the initial object. The icon created will include a generated piece of text, according to the nature of the replaced subexpression (Sum[100], Cos(...) or Matrix[20,20] for example). The expansion may then be achieved directly by spotting with the mouse, or indirectly with the magnifying glass.

- **Local abbreviations**
  The matter is to substitute, locally in a formula, all occurrences of an expression by an abbreviation. The expression corresponds to the marked block. The abbreviation can be typed on the keyboard in a dialogue window or be set to a default value (LOCAL_i if i is the i-th local abbreviation requested for the current formula).

- **Simple local abbreviations**
  They work on the same model as local abbreviations except that only the marked area is substituted by the abbreviation. The default value is : $\mathsf{LOCAL}_i^{\wp}$, the $\wp$ (for partial) symbol records that the substitution was not applied to all occurrences.

- **Global abbreviations**
  The principle is the same as for local abbreviations excepted that the couple abbreviation/content is defined globally, and is then available in each editing window. The default value is ABBREV_i. The index numbering being also common to the whole application.
  Two commands realize global abbreviations : the first one consists in abbreviating the marked block (this defines a new couple abbreviation/content). The second one consists in realizing all possible substitutions of subexpressions according to the abbreviations already defined.
  All the couples corresponding to defined global abbreviations may be displayed in a window by pressing a button on the main control panel.

- **Simple global abbreviations**
  They work the same way as global abbreviations, except that the subject of the substitution is the marked block only. The default value is : $\mathsf{ABBREV}_i^{\wp}$.

- **The magnifying glass**
  The magnifying glass is a graphical feature made for displaying in a separate window the content of a reduction or abbreviation. It is selected by pressing a button on the main control panel and changes the design of the cursor onto a kind of magnifying glass. The user can drag it above an icon corresponding to a reduction or an abbreviation, and click on the mouse button to get a new window with the content of the abbreviation inside it. Such magnifying windows can be either killed at the release of the mouse button, or "pined up" on the screen.

4.3 Global manipulations

- **The scratch paper**
  A window named "scratch paper" is available by clicking a button on the main control panel. Its use is to collect parts of formulae cut by the user in mathematical editors. These parts of formulae may be syntactically incorrect and could be freely manipulated on the scratch paper.

- **Previously marked block manager**
  Most of the advanced interface functionalities use as argument the expression contained in the marked block. For this reason, it would be useful to keep the last marked blocks in memory. Help could be provided by a graphical component managing the current and previously marked blocks. For example, this could be done by a window including on the right the current marked block and on the left a sub-window linked to a vertical scrollbar displaying in five buttons the

five previous marked blocks in an iconic form. Clicking on one of the five buttons would change the current marked blocks. Using the scrollbar would allow the retrieval of the oldest marked blocks. Such a mechanism presents two major advantages : it makes edition easier for the user who can access five permanently updated blocks, and it increases the feeling of security during complex manipulations (a block erased accidentally remains available for a while).

- **Using the history**
  The history mechanism is similar in systems such as MACSYMA or MAPLE and in MATHSCRIBE. It enables the user to retrieve and use in the currently edited formula the previous or the n-th expression given to the CAS. To generalize the use of this facility it seems desirable to bring the following improvements : pattern-matching based research, insertion of comments usable as research keys. More, the history must be connected to a kind of vertical scrollbar.

- **Scrollbars**
  Each mathematical window includes two kinds of scrollbar. The first one is horizontal and is suited for a latteral scan of the formula. The second one is vertical and is linked to the history. It is desirable to let the user chose between connecting the vertical scrollbar to the requests, to the answers or both (like in MATHSCRIBE). In effect when the results are long a scrollbar exclusively connected to the requests is best suitable for a research in the history. In return, retrieving a precise result is easier if the scrollbar is connected to the results only.

- **Comments usage**
  In addition to the comments used in the history mechanism, it may be useful to attach another kind of comments to any object of the CAS. This is particularly true inside an unorganized structure such as the scratch paper. These comments will be a concept local to the interface and will never be transmitted to the CAS. For each elementary lexical unit (as a number or an operator), it will be possible for the user to create, modify and erase comments freely and at any time. The comments will be displayed in a particular font. Furthermore, they could be iconified by clicking on them with the mouse.

4.4 Utilization of a command language

All CAS include a command language which enables one to master all the possibilities of the system. The user interface will have to provide specialized tools to ease the use of this command language.

- **syntactical edition**
  In the mathematical windows the edition of expressions will be directed by the syntax of the command language. This includes the edition of mathematical formulae.
  For the interface this implies a precise knowledge of the syntax of the manipulated form and an abstraction level superior to a simple characters editor. The objective is to apply powerful treatments to the edited text, such as dynamical verification of the syntactical validity. For the user that means that the editor will try for each recognized lexical unit to match the text with a valid syntactical pattern. Thus, the editor will be able to reject unacceptable syntactical units in a given context. It will also be able to insert empty blocks that the user will have to fill. Moreover, the editor will take advantage of its syntactical knowledge to anticipate the user stroke. For instance, after the the character (^) is typed, the cursor will go up and the editor will expect an exponent entry.
  Hence the use of a syntax-directed editor presents numerous advantages : the guaranty of the syntactical validity of the expressions given to the CAS, the automatic indentation of the expressions, the anticipation of the user stroke and the correct positionning of the cursor.

- **syntactical scheme generation**
  Clicking in an edition window will cause menus to pop-up offering sub-formula schemes. These schemes correspond to the different constructions available in the command language. Selecting

a scheme with the mouse will make it appear at the cursor location. Here are for instance the syntactical scheme corresponding to an undefined integral : $\int \boxed{expr}\ \mathbf{d}\ \boxed{var}$
These menus will make the edition of complicated mathematical formulae easier and faster. At the same time they will offer a quick view on all possible constructions of the CAS command language and will constitute a tempting alternative to the lecture of the documentation.
These syntactical schemes will also be automatically generated and displayed during keyboard edition, with the editor identifying the beginning of a known lexical unit.

- **Interactive help for the programmer**
  When programming a system like MACSYMA or SCRATCHPAD II the advanced user has to face a great number of concepts : very long list of available modules, complicated graph of types, existence of thousands of functions, etc. That's why the user interface has to provide adapted tools for quick scanning in the on-line documentation. Many ways are possible : use of pattern matching for the look-up of an identifier name, function name completion, fast access to the technical documentation, etc.
  More, the interface can provide assistance for the use of the debugging facilities of the command language. At the very least it will include a graphic version of *debug* or *trace* commands.

4.5 Usage of the color

Color is helpful for immediate perception of the main components of the interface. It is essential to establish for the whole application, a constant relation between a color and an identical concept of the interface. It is also important to limit the number of colors simultaneously present on the screen. This will allow a faster reading based on the contrast of the colors. When many applications are simultaneously present on the screen, using a uniform subset of colors also makes the quick identification of the windows composing the CAS interface easier. More, the user interface will have to be usable on monochrom screens. That is why the colors will never carry supplementary original information but will be strictly used to emphasize existing concepts.

4.6 Communication with the CAS

All the information about the system must be complete, widely available and easily understandable. In the same way, commanding the system must be simple and handy for an unexperienced user. Menus or control panels, offering all the possibility linked to a particular concept, will be available each time these possibilities can be known in advance.

- **Main control panel**
  A window named "Main control panel", always present on the screen, drives the system. It is used to display essential informations about the System and gathers the buttons corresponding to the main functionalities of the interface. Each button, in addition to the name of the triggered action, includes a complementary graphical indication on the nature of the corresponding effect (instant action, scrolling of a menu, opening of a dialogue box, etc). In the case of a menu or a dialogue box, the current and the default value will be displayed nearby the button on the control panel and will be reminded in the menu or the dialogue box.

- **Global variables of the CAS**
  A window must enable one to visualize and modify all the global variables of the system. This window will come in the form of a chart, each line presenting the following information : name of the variable, current value (editable), value by default and indication on all the possible values (for example : boolean, numerical interval, string of characters, etc).

- **Access to the documentation**
  The documentation of the CAS must be available for consultation at any time and the way to do it must be easy and obvious. To do this the main control panel will include a "HELP" button.

Its effect will be to pop up a menu offering the two following possibilities : to select and display the different sections of the on-line CAS reference manual or to get a specific help by pointing out with the mouse a function or a global variable. The different elements of the interface should also be self-documented. That is why it should be possible to use the "HELP" button to get an information on a precise concept of the interface.

- **Algorithms driving**
  The command language of the CAS must offer the one who programs an algorithm the opportunity to communicate with the user during the execution of the code.
  Some complex algorithms such as Knuth-Bendix or Gröbner' basis, derive some benefits from an information exchange between the system and the user. Thus the program can inform the user of the achieved steps and show intermediate calculation results. This can interest the user who can manually modify some important parameters during the execution. In some other simpler cases the program may only tell the user about the launching of a particular subroutine or warn that the coming calculation may take a long time.
  The interaction between the program and the user can be designed according to two different ways : either as a dialogue exclusively mastered by the calculator (the program keeps the user informed and asks him to take some decisions), or as a driving of the algorithm by the user (the program keeps the user informed by regularly displaying some variable contents or plotting a graph, and the user can at any time impose his options by pressing buttons).
  Concerning the interface, this technique will require a dedicated window linked to the algorithm code execution. Besides, the options chosen by the user during calculation will be memorized in the history. A particular symbol will go with the requests which have generated such choices. This symbol will also be used as a switch to visualize the options corresponding to a request of the history. Concerning the algorithm, the one who codes it, will have to integrate relatively simple instructions into the program allowing the interface to generate the dialogue window. These instructions will indicate the kind of dialogue desired, the information to be displayed and the buttons to create with the corresponding actions.

- **Plotting graphs**
  A button on the main control panel must ease the use of the plotting facilities integrated in the CAS. The effect of clicking this button will be to pop up a dialogue window linked to the plot window where the graph will be displayed. The dialogue window will help the user to point out an equation and the different parameters needed by the plot program.

4.7 External communication

In most cases, the user wants to transfer a formula to another application or a printer. On some occasions, the subject of the transfer can be different : algorithms, session abstracts, hardcopies of a window or of the screen, etc. In all cases, the interface must offer a fast and easy way to realize this kind of manipulations.

Thus, we could have three buttons on the main control panel. The first two buttons would pop two menus onto the screen. Selecting in the first one would assign an output format (TEX, PostScript, C, Fortran, Lisp, etc.), selecting in the second one would define an output channel (file, printer, X Window System cut buffer, etc.). Then, the transmission of the content of the marked block would be done by clicking on the third button. Once selected, the output format and channel would be permanently displayed nearby the transmission buttons.

Beyond these possibilities, we can imagine a mechanism which would enable the user to write and to integrate into the interface a function generating code in a particular output format from an internal reference format, syntactically simple and semantically lightened (a Lisp format for example). This would allow the user to establish a communication between the CAS and an application with a particular input format (a spreadsheet or a data base for instance).

4.8 Help in the writing of scientific texts

This is a particular case of communication between the CAS and an external application (text editor). The aim is to answer to three wishes expressed by many CAS users. The first one is the plain integration to a document of a formula obtained with the help of the CAS. The second one is the insertion of a figure (such as a curve drawn by the CAS in a window or just a screen copy). The third one is the generation of a CAS session abstract. This could work in the following way : when given a relevant sequence of couples request/answer, freely selected in the historics of mathematical windows, the system would automatically generate the abstract (in TEX or PostScript format).

4.9 Relation with the window manager

Under X, the following actions : *move* (the moving of a window), *resize* (enlargement or contraction) and *iconify*, (substitution of a window by a graphical icon), are usually uniformly carried out through the window manager. However, the programmer has to write, if necessary, the non-standard methods to achieve each of these actions on the different kinds of windows which make up the interface. This will be the case for the windows including formulae editors. In all cases, it is important that the CAS interface has a familiar look and feel, and is driven by the user's preferred window manager.

4.10 Parametring the interface

Beyond the possibilities offered by the window manager, a large subset of visual characteristics of the interface might have to be modified by the user. This includes : fonts, colors and default window sizes. Moreover, one should be able to carry out some choices dynamically : the use of one or two windows for the dialogue with the CAS, the criteria of caesura of the too long formulae, etc.

5 Conclusion

The absence of convenient graphical man-machine interface has certainly been a serious obstacle to the broadcasting of CAS in the scientific community. Today, the available powerful workstations and software tools for the interface developers allow the creation of more and more sophisticated graphical interfaces for an acceptable cost in term of developing effort. The main problem with realizing such a user interface for a CAS is to satisfy the desire for quality : respecting the typography and the habits of work, while presenting a set of performant and user-friendly tools.

The aim of this document was to give a preliminary impression of what a CAS user interface could look like in the near future. It does not constitute the specification of the best imaginable interface for a CAS, but just a picture of the needs expressed by the users. In the framework of the Safir project at Inria, it will be used as a model for the specification of the user interface of SISYPHE [6].

This paper also presented a development methodology based on the use of high level software tools. From this point of view, the next step of our work will discover new problems. After evaluating them, the point will be to choose among all the existing software tools adapted to our problem, those which are able to be integrated in a large system, and are efficient enough to produce a useful interface corresponding to the initial specifications.

References

[1] M. Bologna, M. Chesi, S. Mannucci, I. Montanelli, P. Torrigiani, and N. Zuffi. The kernel system of Graspin. In *CASE 87*, Cambridge, MA, 1987.

[2] V. Bouthors and V. Joloboff. Editeur d'interface EGERIE. Rapport interne du centre de recherche bull, Bull, September 1989.

[3] Carnegie Mellon University Software Engineering Institute. SERPENT overview. Technical Report CMU/SEI-89-UG-2, Carnegie Mellon University, August 1989.

[4] J. Coutaz. A layout abstraction for user system design. *ACM SIGCHI*, pages 18–24, January 1985.

[5] P. Franchi-Zanettacci, B. Chabrier, and V. Lextrait. GIGAS: a graphical interface generator for attribute specifications. In *Actes du colloque "Le Génie logiciel et ses Application"*, Toulouse, December 1988.

[6] A. Galligo, J. Grimm, and L. Pottier. The design of SISYPHE : a system for doing symbolic and algebraic computations. In A. Miola, editor, *LNCS 429 DISCO'90*, pages 30–39, Capri, Italy, Avril 1990. Springer-Verlag.

[7] M. Green. The University of Alberta user interface management system. In ACM, editor, *Computer Graphics*, pages 205–213, San Fransisco, July 1985. SIGGRAPH'85.

[8] J. Hullot. SOS interface: Un générateur d'interfaces homme-machine. In *Actes des journées AFCET sur les Langages Orientés Objets*, pages 69–78. Bulletin BIGRE+GLOBULE, January 1986.

[9] Ilog, S.A. *Aïda, environnement de développement d'applications*, March 1989. Version 1.3.

[10] Ilog, S.A. *Masaï, L'outil de développement interactif d'interfaces graphiques*, 1989. Version 1.0.

[11] G. Kahn et al. CENTAUR: the system. In E. Brinksma, G. Scollo, and C. Vissers, editors, *Proc. of 9th IFIP WG6.1*. Intern. Symp. on Protocol Specification, Testing and Verification, 1989.

[12] K. Kimbrough and O. LaMott. *Common Lisp User Interface Environment.* Texas instrument, Inc., February 1988.

[13] M. A. Linton, P. R. Calder, and J. M. Vlissides. InterViews: A C++ graphical interface toolkit. Technical Report CSL-TR-88-358, Stanford University, July 1988.

[14] B. A. Myers. Tools for Creating User Interfaces: An Introduction and Survey. Technical Report CMU-CS-88-107, Carnegie Mellon University, January 1988.

[15] OSF/Motif. *OSF/Motif Programmer's Guide, Programmer's Reference Manual & Style Guide*, Open Software Foundation edition, 1990.

[16] G. R. Pfaff. User interface management systems. In *Proceedings of the workshop on user interface*, Seeheim, FRG, November 1983. Springer.

[17] C. J. Smith and N. M. Soiffer. MathScribe: A User Interface for Computer Algebra Systems. In ACM, editor, *Conference Proceedings of Symsac 86*, pages 7–12, July 1986.

[18] R. R. Swick and T. Weissman. *X Toolkit Athena Widgets*, MIT edition, 1988.

[19] T. Teitelbaum and T. Reps. The Cornell program synthesizer: A syntax directed programming environment. *Communications of the ACM*, 24(9):563–573, September 1981.

[20] Tektronix, Inc. *MathScribe User's Manual Version 1.0*, 1988.

[21] S. Wolfram. *Mathematica, A System for Doing Mathematics by Computer.* Addison-Wesley, 1988.

[22] D. A. Young and P. S. Wang. GI/S: A Graphical User Interface For Symbolic Computation Systems. *J. Symbolic Computation, Academic Press*, 4:365–380, 1987.