TéléMac: A Remote-controlled User Interface Server

Gilbert Gagnon
Jean Vaucher

Département d'Informatique et de Recherche Opérationnelle
Université de Montréal
CP 6128, Succ "A" Montréal, Québec
H3C 3J7
E-Mail: gagnon@iro.udem.cdn

ABSTRACT

Modern software is now expected to be used via sophisticated highly-interactive graphic interfaces. Unfortunately, the development of these interfaces requires much investment in both hardware and programming effort. In the paper, we describe TéléMac, a general-purpose, user interface "server" which runs on a Macintosh™ micro-computer but is remote-controlled via a simple RS-232 connection by an application running on a different host computer. TéléMac provides most of the typical Macintosh features such as multiple windows, mouse selection, graphics, dialogues and voice synthesis. It also implements some high level concepts such as connected graphs (not graphics) above and beyond the Macintosh primitives. Most of all, it was designed to be easy to integrate into application programs.

In the paper, we discuss the design of TéléMac, give examples of its use and outline its implementation.

RESUME

Les logiciels modernes se doivent d'utiliser des interfaces usagers graphiques sophistiquées et hautement interactives. Malheureusement, le développement de ces interfaces demande un grand investissement autant en matériel qu'en effort de programmation. Ce papier décrira TéléMac, un "serveur" d'interfaces usagers générales fonctionnant sur un micro-ordinateur Macintosh™ contrôlé, via un simple lien RS-232, par une application en exécution sur un ordinateur hôte. TéléMac donne accès à la pluspart des possibilités du Macintosh telles fenêtres multiples, sélections avec la souris, objets graphiques, dialogues et synthèse de la voix. Il implante aussi quelques concepts de plus haut niveau, tel réseaux graphiques, par-dessus les primitives Macintosh. Plus important encore, il a été conçut pour faciliter son intégration aux programmes d'applications.

Nous parlerons ici de la conception même de TéléMac, nous donnerons des exemples d'utilisations et tracerons quelques lignes de son implantation.

Introduction

A user interface can be thought to be a control panel from which one can invoke an application's functions, and consult and modify its parameters and data [HULL87] . Just as control panels in airplanes have evolved from simple broomsticks and fuel gauges to fully computerized instruments, user interfaces have also come a long way from line-oriented input and output to more convivial schemes based on windows, icons and menus. Unfortunately, the implementation of such user interfaces is generally a difficult undertaking with major impact on the size and complexity of the interfaced application's code.

Various tools meant to simplify this task are now evolving in the area of Computer Human Interfaces (CHI). Some programming languages include graphic functions. Window Management Systems (WMS) offer windowing facilities through sets of subroutines. There is now also research on "User Interface Management Systems" (UIMS) to provide whole development environments for interfaces [COUT87]. Yet, the task of designing and integrating interfaces into applications remains difficult. Although WMS provide libraries of low and intermediate level routines (window creation, getting mouse coordinates , window redrawing...), a lot of work is still needed to combine all the functions in the required manner. Moreover, WMS are usually tailored to one type of graphic workstations [GETT86, SUN] and interfaces are rarely transportable. Specialized languages built over WMS [BRAC87, MICH87] share their characteristics. UIMS are more powerful and often include specialized languages and editors [HILL86, OLSE86] to specify the appearance and functionality of the interface. Nevertheless, they tend to separate the development of the application from that of the interface to the detriment of the application.

KEYWORDS

User Interfaces, Graphics, Macintosh, Interface Server

The basic idea for TéléMac arose from a class assignment given 3 years ago. For an AI course, students implemented about a dozen "expert systems" in a variety of languages and these were evaluated by their peers. It was painfully obvious that all the systems suffered from poor interfaces and this problem prevented a fair and rapid evaluation of the applications. Furthermore, irrespective of domain of expertise (urinary infection, identification of mushrooms or design of parks), the requirements were roughly similar in all cases: multiple windows, menus, mouse driven editing and graphics.

As a result, we started research on the best ways to implement a system which could provide state of the art graphic interfacing with a minimum of constraints on application programs. In particular, the interface should be easy to program and easy to couple to applications irrespective of the implemention language or hardware. This led to two decisions. The first was to use the Macintosh computer as the support for the interface itself. The second was to implement the interface as a smart terminal controlled remotely via an RS232 link. This eliminates the problems of hardware/software compatibility between the application and its interface; however, it introduces the problem of distributed processing to provide a high bandwidth interaction between the user and the interface whilst reducing the bandwidth required between interface and application.

The system described here is the third version of our interface. The first attempt was based on the use of Apple's MacWorkStation™, a remote-controlled toolbox. MacWorkStation suffered from two major flaws: it was hermetic (and could not be augmented) and it used a sophisticated communication protocol needing a large driver to be linked with the application on the host machine. The second attempt was object-oriented and based on MacApp; this was dropped when a better implementation tool (Lightspeed Pascal) became available. The third version is a complete redesign based on our explicit implementation of object hierarchies.

The paper is structured as follows. First, we present an overview of TéléMac. Then, we illustrate its functionality by presenting two applications: an add-on interface to GENIAL, a natural language interface, and a small interactive graph editor. Finally, important design decisions are discussed and some implementation details are given.

Overview of TéléMac

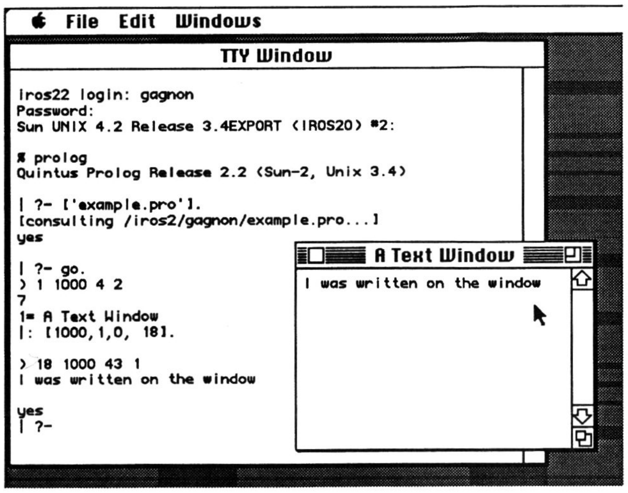
TéléMac starts out by acting as a terminal emulator (Figure 1) with a TTY Window. This allows a user to **logon** to the system of his choice and start an application. The application then communicates with the user via simple read/write instructions as it were addressing an alpha-numeric terminal. However, there is a simple convention which distinguishes simple output which is merely displayed in the TTY window from TéléMac commands which invoke the special interfacing functions: any line which starts with the character ")" is interpreted as a command. The protocol has been deliberately kept simple and all information is transmitted as printable characters.

In the example shown in Figure 1, the application is a Prolog program which creates a window and writes some text into it. The TTY window shows exactly what the application sent to achieve this as well as the replies from TéléMac. The format for messages sent by the host is a one line header

```
<LF> ")" <receiver_id> <sender_id>
          <method_number> <# of parameters>
```

followed by one line per parameter. The format of messages sent by the Macintosh is tailored to the host's application language. The first message which creates the window is from object 1000 (the host) to object 1 (the main object on the Macintosh). Method 4 is the **NEW** message with 2 parameters: in this case, the window type (7) and its title, "A Text Window". The next line:

```
[ 1000 , 1 , 0 , 13 ].
```



```
iros22 login: gagnon
Password:
Sun UNIX 4.2 Release 3.4EXPORT (IROS20) #2:

% prolog
Quintus Prolog Release 2.2 (Sun-2, Unix 3.4)

| ?- ['example.pro'].
[consulting /iros2/gagnon/example.pro...]
yes

| ?- go.
) 1 1000 4 2
7
1= A Text Window
|: [1000,1,0, 18].

) 18 1000 43 1
I was written on the window

yes
| ?-
```

```
I was written on the window
```

Figure 1: Communication example.

is in the format of a Prolog list. It is the reply from object 1 to object 1000; '0' is the code for a **reply** message and 18 is the object number of the newly created window. Thereafter, the application can communicate directly with this window by sending messages to object 18. This is what happens at the bottom of the TTY Window with method 43 which adds text in window 18. To speed up operation and unclutter the screen, it is possible to hide the TTY window.

This small example shows that an application could drive TéléMac with simple standard IO commands (*writeln* , *readln*). Although such operation is possible, it is preferable to go through an interface driver on the application's side to simplify the task. More details will be given later on both the protocol and the driver.

TéléMac's ease of use is partly due its use of the Macintosh *resource* concept. A *resource* is a data structure which describes graphic objects. The Macintosh uses this feature to define windows, fonts, pictures as well as many other things. Resources are created on the Macintosh using resource editors (*Dialog Creator, REdit, ResEdit*...) and kept in libraries called resource files. Applications retrieve these resources by their resource identification numbers. Most often, TéléMac will call upon resources stored locally (i.e. calling dialog #401) instead of downloading the definitions. TéléMac can also manipulate locally stored text files. This is particularly useful with large files of HELP information.

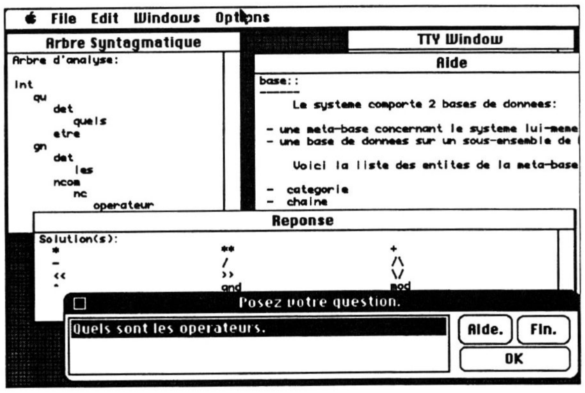
Now we show some applications of TéléMac.

Instrumenting an existing application

Here we describe how a large existing application, GENIAL, was modified to use TéléMac. This was an important test in verifying that a suitably designed interface could provide powerful interfacing with a minimum of extra programming. GENIAL is a natural language database interface system comprising 5000 lines of Prolog [PELL86a, PELL86b]. Initially, Génial was designed for a line-oriented front-end. The user types his request in plain french and the program answers it. GENIAL can also give information on the request's parse tree and semantic transformations. One fundamental problem with this approach is that the information tends to overflow the screen. Another problem is that GENIAL's interactive help system is based on a strict keyword match. In a natural language interface where one may ask a question in many ways, we felt it was too restrictive to ask the user to type syntactically fixed commands to obtain Help [DESA87]. We'll now see how we modified GENIAL to give it a much friendlier user interface.

We had three concerns in the modification process: a) to add code to the application relatively fast, b) the added code should be minimal so as not to obscure the application's code, and c) most of all, the modifications should be useful and practical. In fact, the work took about two weeks including the initial design and implementation of the driver and much work on TéléMac itself. About one hundred lines of Prolog code, excluding the stand-alone driver, were actually added to GENIAL plus a few global modifications (for example changing *writeln*'s for *w_writeln*'s to output text into windows).

The first step was to implement a separate window for each type of output (Help, trace, syntax tree, etc...). As shown in Figure 2, this allows the user to selectively make visible or hide the windows whose output he wanted to see. To simplify the task of redirecting GENIAL's output to windows we implemented routines mimicking those found in Prolog and many other languages (i.e. Prolog's *write* predicate became *w_write*, the first w standing for window). We also supported the concept of a default output window to keep the generic status of GENIAL's low level output routines. To handle request input from the user we used a modeless request asking dialogue which could be pushed aside by the user and brought back to front-screen by GENIAL whenever it was ready to accept input from the user.



```
Arbre d'analyse:

Int
   qu
      det
         quels
      etre
   gn
      det
         les
      ncom
         nc
            operateur
```

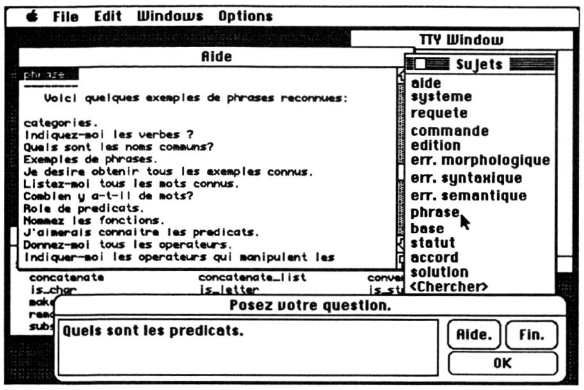
```
base::
------
    Le systeme comporte 2 bases de donnees:

- une meta-base concernant le systeme lui-meme
- une base de donnees sur un sous-ensemble de

    Voici la liste des entites de la meta-base

-  categorie
-  chaine
```

```
Solution(s):
  *              **             +
  -              /              /\
  <<             >>             \/
  ^              and            mod
```

```
Quels sont les operateurs.
```

Figure 2: Windowed GENIAL
Different windows for different information channels

Secondly, GENIAL's HELP system was redesigned to be easier to use with TéléMac (Figure 3). A text editing window (**AIDE**) is created within which a local copy of the HELP file is opened. The user can scroll through the file at will to search for topics. But the addition of a menu dialogue (**Sujets**) with the names of certain attainable topics permits the user to browse faster. On selection of a topic name, a simple repositioning of the text within the window brings the information in sight. A free text search was even implemented to offer more detailed browsing. Text repositioning and searches are done locally on the Macintosh upon request from the host.



phrase

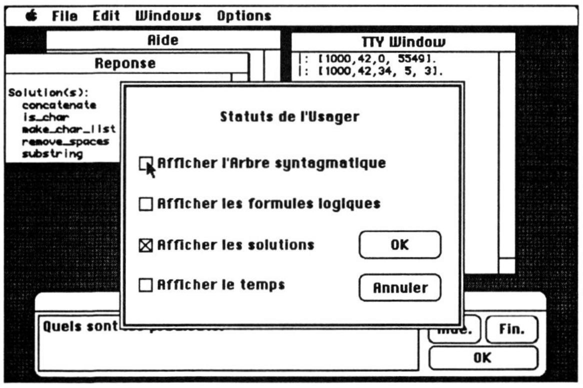
Voici quelques exemples de phrases reconnues:

categories.
Indiquez-moi les verbes ?
Quels sont les noms communs?
Exemples de phrases.
Je desire obtenir tous les exemples connus.
Listez-moi tous les mots connus.
Combien y a-t-il de mots?
Role de predicats.
Nommez les fonctions.
J'aimerais connaitre les predicats.
Donnez-moi tous les operateurs.
Indiquer-moi les operateurs qui manipulent les

Quels sont les predicats.

Figure 3: GENIAL's help system
Note the free text search item "<Chercher>"

Finally, there were some options in GENIAL to indicate, among other things, the kind of output wanted or if a journal of the session should be kept. Those too were redesigned using simple modal dialogs with check boxes to indicate the state of an option. The dialogs were called by selecting an option in GENIAL's menu (the same menu where the help topics dialogue could be requested). Again this was as easy to implement: the application simply invokes a specified dialogue from a Macintosh resource file and analyzes the list of item values returned by TéléMac.



```
|: [1000,42,0, 5549].
|: [1000,42,34, 5, 3].
```

Solution(s):
concatenate
is_char
make_char_list
remove_spaces
substring

Statuts de l'Usager

☐ Afficher l'Arbre syntagmatique
☐ Afficher les formules logiques
☒ Afficher les solutions
☐ Afficher le temps

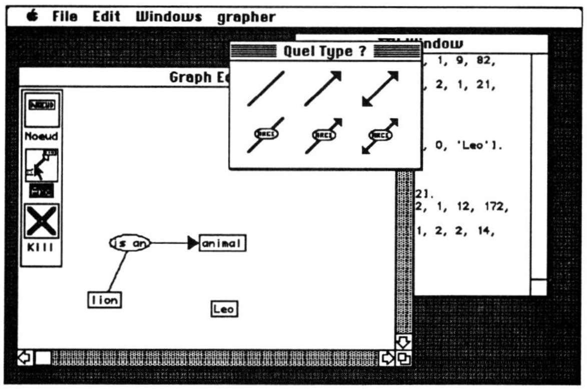
Quels sont

Figure 4: User status dialogue

From graphic objects to graphs

The previous section emphasized the use of multiple text windows, menus and mouse-driven editing. TéléMac also supports direct manipulation of graphic objects. For example, the standard Macintosh primitive figures (rectangles, ovals, lines, icons, etc...) can be drawn remotely. Furthermore, these figures are implemented as local objects with special attributes indicating permissible user actions. The attributes include *active*, *moveable* and *copyable* which specify if an object can be moved or be duplicated as a result of mouse *dragging*. In all cases, the application is informed of such actions by messages from the interface and can react through event handling routines. Objects can be moved by the application to provide a limited form of animation.

A very important application of graphics, and one we wished to support early on, is the display and manipulation of networks or graphs. Already with graphic objects, an application can build such graphs but we expected that users would like to move arcs and nodes at will for a better view. If each graphic element must be controlled individually and remotely, this kind of manipulation requires a lot of information transfer between application and interface. To handle the direct manipulation of networks, we implemented a specialization of the graphic window, the grapher window, whose role is to manage and display graphs. It maintains local information on the graph's connectivity, redraws arcs when the user moves nodes and reports to the application any user actions on the elements (nodes or arcs). To maintain the integrity of the displayed graphs, TéléMac considers arcs and nodes to be special objects: they are not directly accessible from the application, rather, the application must ask the window to create and manipulate them.



```
, 1, 9, 82,
, 2, 1, 21,
, 0, 'Leo'].
21].
2, 1, 12, 172,
1, 2, 2, 14,
```

Figure 5: GraphEdit, a simple interactive graph editor displaying the arc type menu dialogue

At the time we developed the grapher facilities, there was no ready-made application in need of this type of interface. As a result, we decided to build a small program to act as a test-bed. This was GraphEdit, a small interactive graph editor comprising less than a hundred lines of Prolog code (excluding the driver). An abridged version may be found at the end of this paper (Listing 1). The idea was to give the user a set of tools to construct a relational type graph with nodes and possibly tagged arcs. Figure 5 shows the Editor interface. Available tools are represented by an iconic menu (top-left). They include **Node** and **Arc** creation and destruction (**Kill**). A tool is invoked by clicking on its icon. Thereafter, the user is prompted for further information such as names of nodes or types of arcs (shown in Figure 5). New nodes are placed in the middle of the drawing for the user to place where he wants. Arcs are drawn from the clicked source to clicked destination. At any time, the user may move the nodes and its arcs will track the movement automatically. A menu (titled "grapher") is provided with options to save the current graph to a file, to restore a saved one or to quit.

GraphEdit makes good use of resources for dialogue templates and icons. A resource file ('gedit.appl.rsrc') is opened at application initialization time (listing 1, line 1). Then the grapher window is created specifying an event handling routine, *ge_click*. Finally the tools icons and the menu are installed (line 2) before calling the event loop to wait for user actions. It must be understood that graphic objects and graph elements are numbered sequentially from 1 upon creation. The core of the program is the event handling function *ge_click* (line 6). Here click events sent by TéléMac are analyzed and reacted to accordingly. If the action specified is a click or drag (copy) (lines 6 and 7) on one of the tools then a node or arc is created or deleted else the default click routine for graph elements *gr_bid* which maintains the correctness of the local nodes' positions is called (line 8). Simple driver procedures save the graph from this information or restore it from a file (lines 3 and 4).

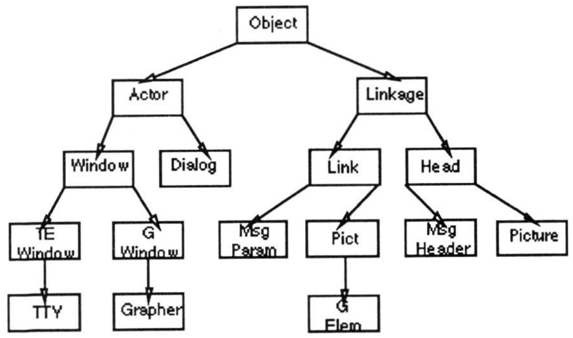


Figure 6: Hierarchical structure of objects within Télémac

Recently, a more powerful editor supporting node and arc attributes for Entity-Relational Modelling has been implemented using TéléMac's grapher facilities.

Inside TéléMac

The Macintosh part of the program is comprised of about 9000 lines of LightSpeed Pascal™ code. Among the first aspects of TéléMac to be established were the syntax of the messages to be exchanged and the fact that the program should be structured along the object paradigm (similar to [JACK87]). The main class of the system is the Object class which serves the purpose of memory allocator for objects. Objects are the substrate for Actors and Linkages. Actors form a SUPERCLASS for all objects directly controllable by the host application. Lists are implemented on the Simula-67 model and Linkage is the SUPERCLASS for Links and Heads. Links and Heads are themselves specialized into message parameters and heads, used in the Macintosh-host communication, and Picts and Pictures. Picts are the primitive graphic objects which are assembled into Pictures owned by graphic windows. Actors come in two categories, Dialogs and Windows. Windows are specialized in two major classes, Text Editing and Graphic. Text Editing Windows not only handle text editing and scrolling but also file manipulation and string searches. Graphic Windows handle the creation, destruction selection and manipulation of the Picts object they own via their Picture. These facilities may be invoked either by the user or the application. There's also the TTY class, a specialization of the text editing window which is responsible, through its only instantiation, the TTY Window, for the communication between the Macintosh and the application at rates between 300 and 9600 bauds. And finally there's the Grapher and the Graph Element classes, the former a descendent of the Graphic Window handling the graph's connectivity, the latter a specialization of the Pict class representing nodes and arcs.

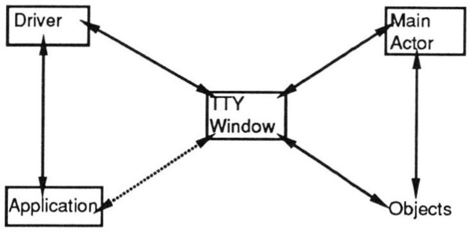


Figure 7: Communication within the TéléMac system

Two object instances also play an important role in TéléMac. We've already mentioned the TTY Window object; the other is the MainActor which handles the creation of new objects by the application and returns their assigned identification number. The Main Actor also has the responsibility of non object facilities like menu allocation and item selection reporting, and speech synthesis.

Communication between the objects is the result of calling one of a series of routines that dispatches the message to the invoked object. Communication from the objects to the host is done by sending messages to the TTY Window object which outputs them to the serial communication line. Messages incoming from the host are handled by the same TTY object, transformed into a message lists and sent to the destination object. The syntax of the external messages, those between the Macintosh and the host, is a problem of upmost importance. The messages to the Macintosh objects have a fixed syntax easy to implement in any programming languages. First a header line is sent followed the parameters each on a separate line. The syntax of each one depending on the invoked method. To ease the task of analyzing incoming messages by the host, their syntax should match the high level input facilities of the language used. Two languages are now supported by TéléMac; Prolog and Lisp.

We stated before that an application could control TéléMac directly via *write* and *read* instructions. Figure 1, showed how it could be done. It also indicates the problems faced with this approach:

— The user should have high-level access to the objects of the interface. The message syntax is quite simple but nonetheless strict. Messages must be built and analyzed carefully even if this syntax is suited to the language used (i.e. lists for Prolog and Lisp).

— To avoid the handling of cryptic messages by the application there should be a database of the objects created and their attributes (i.e. position, name...). The maintenance of such a local model should not be left to the application. In the previous example, the window was referred to by a given name and not its numeral identification.

— User interaction with a graphic interface is asynchronous by nature. User's actions guide the application's execution of the application. For this reason the Macintosh's toolbox is event oriented in its design. There must be an event loop cycling, waiting for events and dispatching them to appropriate handlers. Remote control of the interface only means that these events, or rather their TéléMac treated counterparts, must be sent through the link to the host. There, they may still arrive at any time and the application must receive them. Thus an event loop has to be present in the driver which will receive the TéléMac events from the Macintosh and execute application specified routines to handle them.

These considerations lead us to develop a software interface or "driver" which accepts high-level requests from the application and handles the coded communication on the physical link. The brevity of the listing of the GraphEditor which uses the driver shows the advantages of the "driver" approach. The current Prolog driver is a full-fledged implementation consisting of more than one hundred predicates supporting all remote functionalities of TéléMac on less than one thousand lines of code. For example, two of these predicates, (*create_tewindow* and *w_write*) could be used for the same results as illustrated in figure 1. The diver is modular and only the parts required need be loaded. A programmer's manual describing the system is available [GAGN87].

Conclusions

TéléMac has shown that it is possible to provide a state-of-the-art graphic interface that can be integrated easily into existing applications. TéléMac has been found a useful adjunct to the GENIAL2 natural-language system and is now being used for a conceptual editor and an expert system.

Implementing an interface as a remote-controlled program is useful for two reasons. First, it frees the interface from the necessity to use hardware and software compatible with that of the application. Secondly, the low bandwidth of the link imposes a clean functional decomposition between the application and the interface. This opens the way to development of general-purpose independent interface servers.

The Macintosh proved to be a powerful support for the interface. We made full use of its graphic primitives, toolbox functions and object templates (resources). However, it was necessary to extend these functions, for example with the notion of connected graphs, and to simplify the interface programming by providing higher level operations with many defaults.

TéléMac is a useful bit of software, but it is also an experimental vehicle and it can be expected to evolve. At present, we are studying ways to introduce more parallelism to allow independent processes in the application to communicate asynchronously with their display counterparts in the interface. We are also considering ways to distribute processing by downloading "programs" to be interpreted by the interface.

Acknowledgements

This work was supported in part by the Natural Science and Engineering Research Council of Canada under Operating Grant A7699.

References

[BRAC87] BRACHMAN, B., GProlog User Manual. Department of computer Science, University of British Columbia, Vancouver, 1987.

[COUT87] COUTAZ, J., The construction of User Interfaces and the Object Paradigm. Proc. European Conference of Object-Oriented programming, Paris, Bigre+Globule, n° 54, Juin 1987, Publ. IRISA, Campus de Beaulieu, 35042 Rennes, France, pp.135-144.

[DESA87] DESAI, B., C., FRASSON, C., VAUCHER, J., Integrated Intelligent Interfaces for Office Information Systems, to appear in the proceedings of IEEE Compint '87, Montreal, November 9-13 1987.

[GAGN87] GAGNON, G., VAUCHER, J., TéléMac. Manuel d'Utilisation. Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Publ. N° 191.

[GETT86] GETTYS, J., SCHEILFLER, R. W., The X Window System, ACM Transactions on Graphics, Vol. 5, No. 2, April 1986, pp. 79-109.

[GREE86] GREEN, Mark, A Survey of Three Dialogue Models. ACM Transactions on Graphics, Vol. 5, No. 2, July 1986, pp. 244-275.

[HILL86] HILL, R. D., Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction — the Sassafras UIMS, ACM Transactions on Graphics, Vol. 5, No. 3, July 1986, pp. 179-210.

[HULL87] HULLOT, J.M., SOS Interface, un générateur d'interfaces homme-machine. Actes des Journées AFCET: Langages Orientés Objet, Bigre+Globule, 48, Publ. IRISA, Campus de Beaulieu, 35042 Rennes, France, pp.69-78.

[JACK87] JACKY, J. P., KALET, I. J., An Object-Oriented Programming Discipline For Standard Pascal, Communications of the ACM, Vol. 30, No. 9, September 1987, pp.772-776.

[MICH87] MICHARD, A., MONCEYRON, E., Le Système Graphique ASH-Prolog et Son Utilisation Pour le Prototypage Rapide d'Interface Homme-Machine, INRIA-Centre de Sophia-Antipolis.

[OLSE86] OLSEN, D., R. JR., MIKE: The Menu Interaction Kontrol Environment, ACM Transactions on Graphics, Vol. 5, No. 4, October 1986, pp. 318-344.

[PELL86a] PELLETIER, B., VAUCHER, J., GENIAL: Un Générateur d'Interface en Langue Naturelle, Proceedings, Sixth Canadian Conference on Artificial Intelligence, Presses de l'Université du Québec, Montréal, 1986, pp. 235-239.

[PELL86b] PELLETIER, B. Système d'Intérrogation de Banque de Données en Langue Naturelle, Document de travail #177, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, mars 1986.

[SUN] Sun Microsystem, NeWS Technical Report.

Listing 1. GraphEdit

```
(1) graphedit :-
        init_interface('gedit.appl.rsrc'),
        create_grapher( ge, 'Graph Editor', docZW, [150, 50, 500, 300],
                        default, default, ge_click),
(2)     add_picture( ge, icon(7, 441, [5, 5], 'Noeud' ) ),
        add_picture( ge, icon( 1, 442, [5, 55], 'Arc' ) ),
        add_picture( ge, icon( 1, 443, [5, 105],'Kill' ) ),
        add_picture( ge, framerect( 0, 1, [2, 2, 40, 162] )),
        install_menu( gemenu, 5, 'grapher', ['Save To File...','Load From File...','(-','Quit'],
                      [geSave, geLoad, m_bid, geQuit] ),
        event_loop.


/* handles save menu request by saving the graph on file */
(3) geSave( Wid, Item ) :-
        get_modal_dialog( 400, [1, _, _, Text | R] ),   %% True if button 1, OK, was pressed
        graph_to_file( ge, Text ).

/* handles load menu request by  getting a graph from a file */
(4) geLoad( Wid, Item ) :-
        get_modal_dialog( 400, [1, _, _, Text | R] ),   %% ask for a file name
        clear_window( ge ),                             %% destroys the  window's contents
        ...     %% reinstall icon tools ( 4 lines )
        file_to_graph( ge, Text ).

/* handles quit menu request and return to Prolog's top level */
(5) geQuit( Wid, Item ) :-
        remove_menu( gemenu ),
        kill_window( ge ),
        finish_event_loop.                              %% terminates the event loop

/* handles mouse events within the grapher window */
   ge_click( GRid, Parms ) :-
        get_gwindow( GRname, GRid, _ ),
(6)     ( interface_g_action( select, Parms, NbClick, ObjID, X, Y ) -> %% an object was selected
            ( ObjID = 1 ->              %% the add node icon was hit
                get_modal_dialog( 400, [1, _, _, Text | R] ),
                add_node( GRname, rect, 40, 40, Text)
            ; ObjID = 2 ->              %% the add arc icon was hit
                get_menu_dialog( 401, Hit ),        %% ask for arc type with iconic dialog
                ( Hit >= 4, Hit =< 6 ->             %% if tagged, ask for a name
                        get_modal_dialog( 400, [1, _, _, Text | R] ) ; Text = '' ),
                get_rubber_line( GRname, X1, Y1, X2, Y2 ),
                ...     %% find the nodes ObjID1, ObjID2, under the line ends ( 5 lines )
                ArcType is Hit + 3,
                add_arc( GRname, ArcType, ObjID1, ObjID2, Text )
            ; ObjID = 3 ->              %% the kill icon was hit
                get_rubber_line( GRname, X1, Y1, X2, Y2 ),
                find_pict( GRname, X1, Y1, ObjID1 ),
                ( get_node( GRname, _, _, _, _, _, ObjID1 ) ->          %% is it a node?
                        del_node( GRname, ObjID1 )
                 ; get_arc( GRname, _, _, _, _, ObjID1 ) ->             %% is it an arc?
                        del_arc( GRname, ObjID1 ) ) )
(7)     ; interface_g_action( copy, Parms, NbClick, 1, NewObjID, X, Y ), %% the node icon was copied
            ...     %% delete the copy and add the new node  ( 3 lines )
(8)     ; gr_bid( GRid, Parms ) ).  %% driver's default handling routine to update the local connectivity
```