ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S13MMI04

The Elettra Man-Machine Interface

Franco Potepan, Giuseppe Surace, Marco Mignacco
Sincrotrone Trieste
Padriciano 99, 34012 Trieste, Italy

Abstract

ELETTRA is a third generation Synchrotron Light Source under construction in Trieste (Italy), with beam energies between 1.5 and 2 GeV. Two networks connect three layers of computers in a fully distributed architecture. An ergonomic and unified approach in the realization of the human interface for the ELETTRA storage ring has led to the adoption of artificial reality criteria for the definition of the system synoptic representation and user interaction. Users can navigate inside a graphic database of the whole system and interactively edit specific virtual control panels to operate on the controlled equipment. UNIX workstations with extended graphic capabilities as operator consoles are used in the implementation of the PSI (Programmable Synoptic Interface), that was developed on top of X11 and PHIGS standards.

I. Introduction

We may think of an ideal man-machine interface (MMI) as a tool which allows users to interactively specify their preferred means of interaction with the devices controlled, using graphic programming techniques to compose predefined objects that support the exchange of numeric, string or graphic information. A definition of new specific objects is also possible using an editor with similar properties.

Commercial interfaces ado not generally combine these two requirements without the introduction of a considerable amount of new concepts and notions. Today we can profit by the wide diffusion of some advanced graphic user interfaces (GUIs), based on a well defined set of composable objects, or *widgets*, that have certainly increased the knowledge of interface principles and interaction paradigms among users. A similar awareness at the lower level, where graphic primitives and basic interaction techniques should be structured together to form an object, is at present unthinkable. As a matter of fact, a global definition of the semantic and syntax of graphic human-computer interaction still suffers from a lack of standardization of the relative graphic and dialogue lexicon, which is a young field of active research. Interactive widgets composers are indeed filling up the market, while a similar approach for the definition of widgets is practically abandoned.

II. Design Goals

In our project of a MMI for the Elettra storage ring the aim of obtaining a well balanced integration between new powerful features available on modern hardware and software platforms, and a comprehensive exploitment of innovative methodologies concerning human-machine interfaces, has led to a continuous revision of the design and implementation phases, together with a considerable effort in testing prototypes with real users.

The usual top-down design approach which uniquely guarantees consistency in terms of colour coding, menus and dialogues layout, warning messages and overall behaviour of the user interface is left at the end, when all possible user interactions are already defined.

Two main lines have been followed in the design of Ψ (Programmable Synoptic Interface), in order to keep the amount of new concepts required for its operation to a minimum:

- to hide all the details concerning the specific structure of our control system from the users. A complete transparency of operative system, programming language, graphic and communication libraries is therefore essential.

- to take full advantage of all the notions users are already familiar with: the local operations of the devices, the planimetrical layout of the machine, the commonly used desktop metaphor as a computer interface.

III. Conceptual Organization

A possible solution to these demands was to adopt typical artificial reality criteria for the design of our interface. A planimetric representation of the whole system, where all the items controlled are shown with their real shape and position, was recognized as the users' most familiar environment. The equipment displayed inside the environment is associated to virtual control panels which allow users to operate the graphic representation of a large set of devices like switches, knobs, sliders, digital and analogue indicators, whose behaviour is equivalent to that of the same instrumental devices. A set of well defined interaction paradigms regulates the navigation inside the environment, the modification of the scale and visibility of the layers into which the graphic information is structured, the selection of devices and the activation of the relative control panels.

Let us now distinguish between the principal elements that compose our artificial world:

-an *environment*, i.e. a synoptic representation of the whole system;

- a set of *objects*, i.e. the devices controlled;

- a collection of *virtual control panels*, i.e. the logical grouping of controls associated to one or more devices;

- a set of *interaction paradigms* between the user, the environment and the objects.

The environment consists of a complete graphic database of the whole system, no longer restricted to fixed size views of selected parts of the plant. If we define the graphic database at system level, freeing the user from the synoptic drafting phase, we can profit by the centralized organization of the data, exploit all the advanced features of the adopted graphic system and make graphic conventions and interactions rules consistent and comprehensible.

Objects are the graphic representation of the controlled devices. They retain, as far as possible, their original planimetrical shape and location, without any additional graphic coding. They are quite similar to the other graphic primitives inside the environment, like buildings and planimetrical references, but they can also be selected and activated, and their representation can be changed according to their state.

Each object is associated to one or more control panels, within which users can operate on the virtual controls to modify the values of a specific controlled device.

Interaction paradigms vary between the different parts. Users can interactively navigate inside the environment, specifying the position and the size of the view. A hierarchical organization into layers of the visualized data and a well defined set of properties can easily overcome the high density of information inside the environment. Layers can be selectively visualized, while further detail and information in text form can be automatically obtained as a threshold function of the scale; logically related objects can be highlighted or changed in scale to improve readability.

IV. Implementation

The Elettra Control System has a fully distributed architecture, organized into three layers of computers: control room, local process computers and equipment interface units. Two networks interconnect the adjacent computer layers. At control room level UNIX workstations are used as operators' consoles [1]. This architecture, however highly adaptive to future developments, implies the adoption of heterogeneous hardware and software between the two networks. Whereas the transparency of the lower operative system is guaranteed by the Remote Procedure Call application protocol [2], naming conventions and servers location among the process level computers are still required at console level.

We have adopted the X Window System Version 11 as our windowing system and basic graphic library, together with the OSF/Motif window manager, Xt and Xm libraries [3][4]. The Motif widget set had to be extended with several home-made widgets to fulfil typical accelerator control requirements. The wide application of this GUI to accelerator controls has led to an active collaboration and exchange of information between several institutes. This might be a good starting point for a definition of common requirements.

Although X11 includes a reasonable number of routines for creating basic graphic primitives and for modifying attributes, it is definitely limited as far as display-list organization and mass storage is concerned. It is mainly a bit-mapped oriented library and the download of visualized portions of graphs is the only storage method available. The lack of any form of data structure within a display list in memory and the consequent impossibility of recognizing a single graphic primitive within the display, of rotating, translating or resizing it, confirmed the inadequacy of this library for the synoptic representation of our storage ring.

We therefore opted for the adoption of PHIGS (Programmers' Hierarchical Interactive Graphics Library) for vectorial graphics. Actual implementations of this standard (ISO-IS in 1988) move towards its complete integration into the X11 graphic interface and event handling merging [5][6]. The PEX (Phigs Extended X) extension of the X protocol, now appearing in its first implementations, provides the transfer of both bit-mapped and vectorial primitives over the network. The advanced display list organization of Phigs into a hierarchical tree structure of primitives [7] perfectly matches our requirements for the centralized database. Moreover, this standard supports unique definition of primitive attributes that includes transformation matrices, in addition to all common vectorial libraries features like archiving and single primitive picking. Maximum performance in different hardware platforms is possible as all primitives suitable of acceleration are downloaded to the graphic subsystem, as specified in the standard. Nowadays, typical graphic acceleration of 1M vectors/s can be easily gained with no optional graphic subsystem. The constant trend of hardware makers towards the adoption of powerful 2D or even 3D graphic accelerators as the default configuration of their workstations, guarantees the acquisition of advanced graphic features, such as real time navigation over a graphic database, with no additional cost.

A. The graphic database

The naming convention adopted in our system is both simple and effective. Any device is defined by its four component string, which specifies the *family* to which the device belongs, the *member* within the family, the type of *action* requested (current read, fault reset, etc.), and the *mode* in which data is transferred (read real, write integer array, etc.) [8].

The main effort in the definition of an exact representation of the storage ring was actually the drafting of all the components in real size through a graphic editor. As this task is always accomplished using common CAD programs, and the machine assembly needs an identical database, we thought of importing the data from our CAD stations via a filtered download of the graphic primitives. The only modification to the CAD database was the addition of a label to each object, specifying its family and member. The link to the control system database was therefore obtained. We imported the pre-existent organization into layers, providing a logical separation between the various parts of the machine, and added the hierarchical structure respecting our naming convention, inserting a further "group" node between layers and families.

We developed a parser that translates Autocad DXF format files and generates a Phigs CSS (Centralized Structure Store). This parser recognizes the notion of blocks and layers and generates the tree structure shown in fig. 1. Buildings and other non-blocked graphic information are also imported and distinguished from the active objects by the absence of a naming label. The resulting structure is a tree of deepness five, ordered in root, layers, groups, families, members and objects.

We tested the good integration of Phigs into X11 defining a widget which loads a generic graphic structure from a file, and provides navigation and primitive picking with no other parameters than the filename inside the Xt widget creation call.

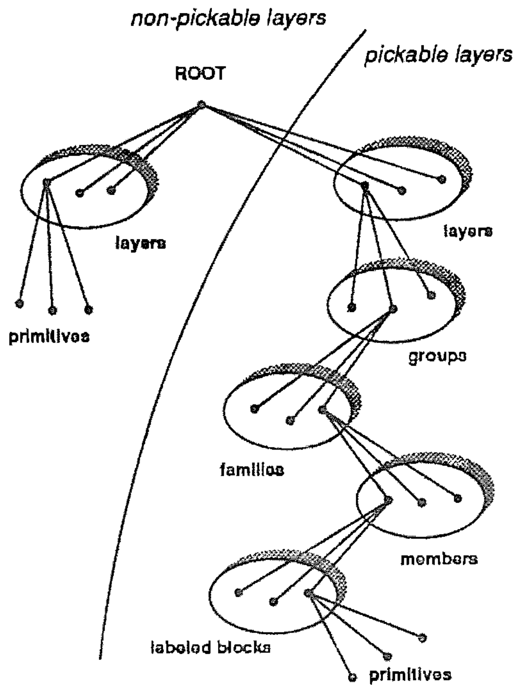

Figure 1. Hierarchical organization of graphic primitives inside the environment.

B. The Control Panel Editor

The time spent in the generation of the code required for a typical composition of widgets inside a control panel was estimated to be excessive when related to the number of panels to be provided to the users. A fundamental problem was also the definition of the types and positions of the widgets chosen for the control of a specified device. The possibility of adopting further application packages on top of those above listed has been carefully investigated. Several programs have therefore been examined in our and in other institutes [9]. Our opinion is that the integration with the underlaying standards is not yet acceptable, with the exception of widget-based interface generators, in which external data connection and home-made widget inclusion is still quite hard. As a result we decided to invest our time in the development of a fully interactive Control Panel Editor (CPE).

We shall distinguish two operative modes from now forward: an edit mode, in which we use CPE to create or modify the panels associated to an object; a run mode, in which we select an object, activate the respective panel and act on the correspondent controls (fig. 2).

The basic interaction tasks (BITs) [10] used in selecting, positioning and resizing widgets inside a control panel are strictly those specified in the OSF/Motif Style Guide document, and are almost identical to those used in other well known GUIs with the same desktop metaphor. The possibility of interactively change selected properties of the widgets that are being edited allows users to have an immediate feedback as the property is changed. We deliberately limited the number of modifiable properties to a definable set, avoiding the change of those properties with an explicit coding, like colour, and of those with an unwanted effect to the interface, or an unknown meaning to the user.

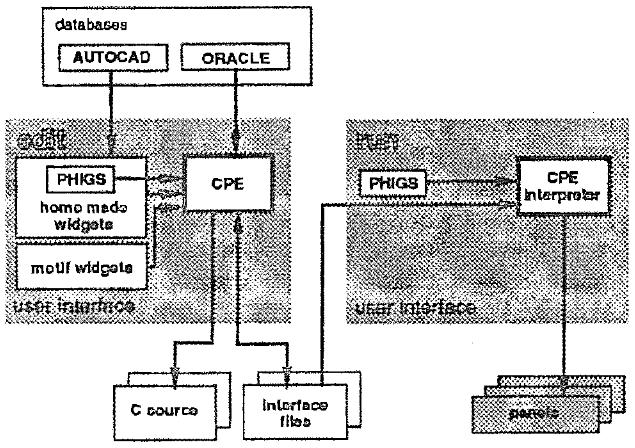

Figure 2. Schematic layout of the Control Panel Editor.

A complete transparency even to naming convention and server location of Ψ was possible with the integration of CPE with the graphic database. Introducing the notion of *association*, which is the only complex interaction task in our interface, we are able to specify a complete data connection with a device.

Let us follow the steps required to create an association. While in edit mode, we select the object we want to control from the synoptic. As the object is picked, its family and member information is used to access a list of all possible action and modes from the machine database. The selection of one of these items highlights all the widgets that are able to handle the specified type of data. As we place the chosen widget inside a panel, the association between the selected device and the widget is defined, and a connection to the relative equipment is established as soon as we enter the run mode and pop-up the panel.

Internally, CPE generates an interface file, listing all interactively selected properties in a purely declarative language. Interface files can be edited with a traditional text editor, enabling system programmers to quickly configurate Ψ with prototyped panels created cutting and pasting properties among interface files.

The generation of pure Motif C source code after panel composition is already available: a self-contained main program is created with explicit callbacks for each edited widget. Application programmers thus have an interactive tool for the rapid prototyping of the graphic and data connection parts of their programs. The implementation inside CPE of a small C interpreter, limited to the recognition of blocks of code, could overcome the limitation of the one-way use of the editor as a C code generator, and guarantees a complete development environment for application programmers.

V. Conclusions

The application of artificial reality criteria to controls is by no means original. Our main effort has been the implementation of these concepts on top of a standard software platform, obtaining a high level of portability among different hardware configurations. The modularity of the various parts of our interface (synoptic, panels, editor) and their consequent easiness of maintenance and upgrade, assures a longer life of the system, compared to the adoption of proprietary solutions.

VI. Acknowledgements

We gratefully acknowledge the support of all the other members of our group: L. Barbina, D. Bulfone and P. Michelini. We are indebted to Fabio Barbo for his valuable work in the definition of the whole graphic database. In particular we wish to thank Claudio Scafuri for his significant contribution to the design and development of our interface.

VII. References

[1] M. Mignacco,*"The ELETTRA Control System"*, NIM A293(1990), pp. 44-49.
[2] P. Andersen et al., *"User Guide to the Network Compiler Remote Procedure Call"*, LEP Controls Note 97, CERN 1989.
[3] A. Nye, T. O'Reilly. X11 Release 4, vol. 1-5, O'Reilly & Associates Inc, 1990.
[4] OSF/Motif Style Guide / Programmer's Guide, Prentice Hall, 1990.
[5] HP-PHIGS Version 2.11 Reference Manual, Hewlett-Packard USA, Jan. 1991.
[6] TGS Figaro+ Rel. 3.0 Reference Manual, Liant Software Corporation, San Diego, Jun. 1991.
[7] T. L. J. Howard et al., *"A Practical Introduction to PHIGS and PHIGSplus"*, Addison Wesley, 1991.
[8] M. C. Crowley-Milling, *"Naming Conventions for the ELETTRA Components"*, Sincrotrone Trieste, ST/M-TN-88/13, Sep. 1988.
[9] Di Maio et al., *Report on the study group for the selection of a software package for synoptics*, CERN Internal Report, Oct. 1990.
[10] J. Foley, A. van Dam, *Computer Graphics principles and practice*, Addison-Wesley, 1990.