DAΦNE TECHNICAL NOTE

INFN - LNF, Accelerator Division

Frascati, Feb. 3, 1992

Note: **C-5**

GRAPHIC USER INTERFACE

Alessandro Stecchi

1. What is a GUI or better a *true* GUI?

When you have a computer in front of you the problem is how to get the incoming information and how to communicate with it. The User Interface is the method for solving this problem. When some kind of graphical representation is involved in this method we have a GUI that is a Graphic User Interface. Actually a real GUI requires the graphic to be interactive. A user must be able to handle windows and not only to watch them, icons, controls and in general any sort of representation shown on his screen.

Many new ideas have contributed to the present way of building a GUI and it is possible to extract some common principles:

- *Icons & graphics instead of text*

We live in a world full of icons that everybody can easily recognize. Men/Women pictures on doors, crossed cigarettes for no smoking, people running away from flames for escape exits and so on. It is an evidence that icons have completely won on text-explanations in a wide context. Icons can be immediate, attractive, concise and cross-cultural. So why shouldn't we use icons for mapping real objects on the computer screen?

- *Metaphors from the real world*

The use of metaphors based on real-world counterparts is widely diffused in modern GUIs. Dealing with metaphors the users have a set of expectations to apply to the computer environment and they operate in an artificial reality.

- *Direct manipulation*

Users want to feel that they are in charge of the computer's activities. They expect their physical actions to have physical results, and they want their tools to provide feedback.

- *See-and-point (instead of remember-and-type)*

Users select actions from alternatives presented on the screen. They rely on recognition, not recall; they shouldn't have to remember anything the computer already knows. Most programmers have no trouble working with a command-line interface that requires memorization, but *the average user is not a programmer*.

Anyhow all those principles tend to the main target of a *true* GUI that is: to hide as much as possible the complexity of an operation. The general form of user actions must be something like "Hey, you . . . do this".

2. Why do we think it is useful to implement a GUI in the DANTE Control System?

The DANTE (DAΦNE New Tools Environment) Control System architecture can be roughly described by a three level scheme:

- PARADISE: the first level where people interact through consoles with the System;

- PURGATORY: the second level where a central CPU works as communication manager and a continuously updated central memory contains all of the machine's variables;

- HELL: the third level where a distributed hardware, resident on several VME crates, performs low level tasks such as A/D D/A conversions, digital I/O, linking toward other standards (GPIB, RS232, etc.) and so on. At this level several CPUs run various applications to control all the machine's elements.

The three levels communicate through high speed channels (parallel busses and optical fibres) via mailboxes where they put and get commands, messages and values. This obviously means that PARADISE must obey a fixed-format syntax in sending a command and it receives plenty of rather cryptic system messages incoming from the lower levels.

Now think of the end user: he probably will be anyone but a programmer and he doesn't want to worry about the computer procedure because he has to worry about the machine one.

For such a person it would be useful to move a knob on his screen with the mouse instead of typing something like:

SET <TheElementName>_<TheFinalValue>_<TheStep>

or turn off a switch by a "click" instead of:

ONLINE <TheElementName>_<[ON][OFF]>

In the same way a front end window containing analog meters, LEDs, alert icons, plots and so on (besides numbers) will be much more intuitive and simple to watch and to be interpreted than a text-based read-out.

A Control System is one of those situations that require an extensive Man-machine interaction and the implementation of a GUI looks very appropriate.

3. What is the effort estimate (in terms of time and code) for the development of a GUI?

What does it mean to write a number into a window? It obviously depends on which number we are speaking about and how we get the window. A Control System involves mainly real-time processes and this means that one has to deal with many short and fast calculations for getting readout or setting values. In other words the quantity of code to write in order to get the temperature of an element is small enough.

And what about a window? Here one has the opposite problem. He must produce a long and slow chunk of code to allocate memory, make all the calls to the low-level graphics procedures and so on.

The ratio between the two jobs is strongly unbalanced and it will raise much more if we think of putting buttons, scrolling text areas, and handling graphics in our window. Considering a complex presentation which involves many windows and menus we can say that a GUI is a good thing but hard to get.

4. Which tools are more appropriate for the development of a GUI (in relation to the scheduled time)?

The general philosophy adopted by the control group is *to use commercial tools as much as possible* for the good reason that a widely diffused software package contains fewer bugs than a new one. There are many tools on the market that automatically generate code in any language (i.e. Prototyper™) and they help you to produce the framework of your application. If you intend to draw and manage windows, buttons, and so on, using such tools you can save your time and drop the number of bugs in the resulting code.

This is a good method for building a GUI but there are some drawbacks:

- You have to write by hand all the subroutines for the display of plots, analog meters, knobs etc. you intend to use;

- Whenever you need to add a control or even slightly move an existing one on an old window, you need to recompile a lot of code.

Whether this can be done or not depends on how much *manpower* you have available. A solution for those problems is to adopt a full environment instead of using a development tool. In this way you can develop and run your application within the same context avoiding the edit-compile-link procedure.

An example is the creation of a window with a button that does nothing on it, using Prototyper™3.0 or a full environment (LabVIEW® 2):

- In the first case you get more than 30 files for a total of more than 2400 lines of code to compile and link;

- In the second one you have only to do a couple of mouse actions (one for "make a new window" and one for "make a new button") and you are ready to run.

The Control System front-end should be easy to update to match the new users requests and to fit controls for the new devices. This comes absolutely natural using LabVIEW® where you have the whole data structure graphically represented in a diagram. You can get a new indicator for a certain value simply by connecting an icon to the related wire in the diagram.

In fact all the functionality of the instruments created with LabVIEW® is due to an icon-based programming language called G. This allows you not only to create a GUI but also to implement a flexible high level software for a first *realtime* processing of your data.

5. An example of control.

A first attempt to implement a control using LabVIEW® is shown. The commission was to build a comfortable way for moving at the same time a cluster of machine elements taking into account a set of corresponding weights. The list of element weights must be editable or resumable from a file. In this control it is possible to create interactively a personalized list of elements by pop-up menus or by browsing the element families. A different window acts as weight editor and automatically fits those in the main panel. (A screen capture of the control is shown in fig. 1).

After setting the step width one sends, via two buttons, a bunch of commands with the calculated values to the lower levels. The composition of the commands and the calculation of each value as well is completely trasparent to the user. The planning and building of this control has taken about 3 days for 1 person and this is a very good number taking into account the complexity of the presentation involved.

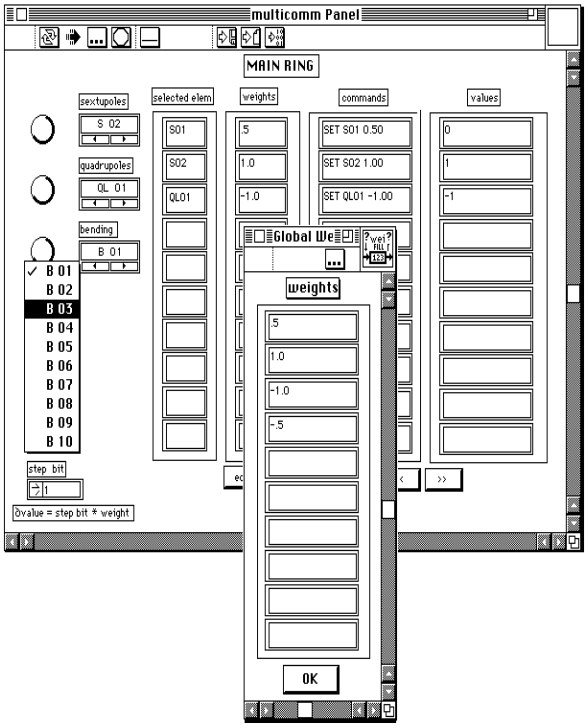

Fig. 1 - Multicomm: an example of GUI obtained with a programming environment.

The control shown in Fig. 1, although very simple has been developed in order to stress the guidelines that we intend to follow in the development of our GUI:

- To use full graphics instead of command-lines. This choice will lead to a strong reduction of errors in typing.

- To allows the user to customize his own instruments at different levels: the first one (like in multicomm) consists in configuring an existing instrument. The second one consists in building a new control starting from a predefined set of commands and elements.

- To implement multiple windows both for command and for read out. One of the most appreciable aspect of LabVIEW® is the possibility of managing more windows at the same time. This means that not only the user can switch from one "panel" to another one but also that the underlaying programs run concurrently. Thanks to this multitasking it is possible a parallel processing and display of the information incoming from the lower levels.