Minimal Requirements for a Graphical User Interface for InfoMall Applications

Gregor von Laszewski

Northeast Parallel Architectures Center
at
Syracuse University
Science and Technology Center
111 College Place
Syracuse, NY 13244-410

1 Introduction

Many applications which will be developed at NPAC in the next years do require some kind of graphical interface to it in order to give a non computer expert the chance to explore different parameter settings for the application[**?**, **?**]. Many users will have experience using PC's and Workstations and know how to use their graphical desktop.
It is desirable to develop a program library providing the applications developer with an easy set of functions in order to make the development of a high quality graphical user interface as easy as possible. This is especially from importance to decrease the high start up time while learning sophisticated graphics packages.
To summarize: We have the goals in mind

- to have a high quality user interface which is attractive and easy to use by the user.
- the functionality of the interface is simple and transparent to the program developer.

2 Design Goals

From the authors experience with different graphical users interfaces and packages including Motif, X11, AVS[2], SUIT[5], Tcl/Tk[4], and XF[3] it takes a long time to write a program. Different strategies might be useful in order to make the one or the other packages easier to use.
The argument it is best to learn from example can not be a sufficient answer to am applications developer dealing with ten thousands of lines. His time is limited and he will not have the time to spend reading extensive documentations.
While using the above mentioned packages for a limited number of applications a small number of graphical objects are obvious to implement supporting the graphics development. This set is the highest level of a hierarchy of libraries. In this way interfaces to different graphics libraries are possible and the best underlying kernel might be chosen for the application in case the standard library should not provide enough functionality.
In case the functionality of the library should be to small the applications programmer can request to extend the library.
To summarize, we request from the library that it is

- simple to use,
- portable,
- and extendable.

3 Graphical Objects

In this section we describe the graphical objects necessary to fulfill the above mentioned goals.

We distinguish two classe of applications. The first class uses a graphical user interface to obtain certain parameters used by the application program. Therfore, the two programs can be developed indipendently. The GUI program calls the apllication program with the help of a UNIX systems command and provides the appropiate parameters necessary to run the program.
The controll flow looks like:

GUI $\longrightarrow$ program $\longrightarrow$ output

Interaction to the GUI is provided when the applipaction program gives the controll back to the GUI. The GUI can use the data provided by the application to update the information on the screen.

GUI $\longrightarrow$ program $\longrightarrow$ GUI $\longrightarrow$ output

Many applications while need the graphical objects as displayed in Figure 1. We distinguish the objects into two classes, namely the

- input objects
- and the output objects.

3.1 Interface

The interface is very simple. Each object is described by its class, its layout and an associated procedure which is executes if the object becomes active.
The general outline is

```
void object_class (OBJECT* object_name,
                           type1 parameter1,
                           type2 parameter2,
                           ...,
                           typen parametern,
                           (void*) action_proc)
```

3.2 Window Manager

The window manager controls the size of the graphical user interface. it is specified by

```
void NPACinitGraphic (int x, int y,
                      int width, int height,
                      char *label)
```

The procedure

```
void NPACexitGraphic ()
```

deletes the GUI from the application and gives the control back to the application.

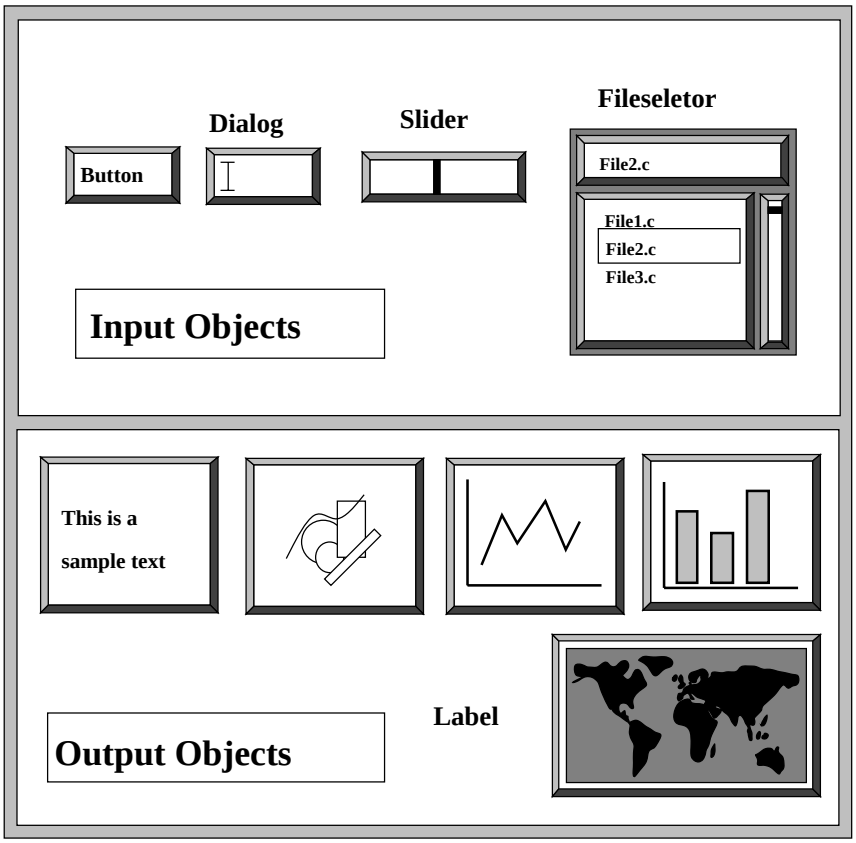

Figure 1: The essential graphic objects

3.3 Input Objects

3.3.1 Button

```
void NPACbutton (OBJECT* object_name,
                 int x, int y,
                 int width, int height,
                 char* label,
                 (void*) action_proc)
```

An object of class button can be placed in the window at position (x,y) with the specified width and height. The font is predefined so that the applications programmer has not to worry about it. If the width or the height parameter are negative the width/height will be automatically adapted so that the button label will fit in the box.

3.3.2 Dialog

```
void NPACdialog (OBJECT* object_name,
                 int x, int y,
```

```
                                int width, int height,
                                char* label,
 char* input,
                                (void*) action_proc)
```

An object of class dialog can be placed in the window at position (x,y) with the specified width and height. The font is predefined so that the applications programmer has not to worry about it. If the width or the height parameter are negative the width/height will be automatically adapted so that the button label will fit in the box. The textual input value is returned in the parameter `input`.
To make the input of different datatypes easier we include the following routines:

```
NPACgetInt    (..., int*    input, ...)
NPACgetFloat  (..., float*  input, ...)
PACGgetDouble (..., double* input, ...)
NPACgetText   (..., char*   input, ...)
```

They distinguish each other only in the type of the input variable. Naturally an input should be checked upon syntax errors. With label a text can be written in-front of the dialog box.

3.3.3 Slider

```
void NPACslider... (OBJECT* object_name,
                                int x, int y,
                                int width, int height,

 type* input,
                                (void*) action_proc)
```

An object of class slider is in the same way placed as an object of class dialog. As with the dialog objects different datatypes are supported:

```
NPACsliderInt    (..., int*    input, ...)
NPACsliderFloat  (..., float*  input, ...)
NPACsliderDouble (..., double* input, ...)
```

3.3.4 Fileselector

```
void NPACdialog (OBJECT* object_name,
                                int x, int y,
                                int width, int height,
                                char* label,
 char* filename,
 char* path,
```

```
                                (void*) action_proc)
```

An object of class file is in the same way placed as an object of class dialog. After selecting a file the variable `filename` contains the name of the file and `path` contains the absolute path of the file.

3.4 Output Objects

3.4.1 Label

```
        void NPAClabel (OBJECT* object_name,
                                int x, int y,
                                char* label)
```

The object label places a textual label on the given position on the window:

3.4.2 Text

```
        void NPACtext (OBJECT* object_name,
                                int x, int y,
                                int width, int height,
                                char* label,
 char* filename,
 char* path,
                                (void*) action_proc)
```

The object text places a textual window on the screen at the given (x,y) location with the specified width and height. The text `text` is written in the object.
The procedure

```
        void NPACrefreshText}  (OBJECT* object_name,
                                     char* text)
```

overwrites the existing text of a text object with given name.

3.4.3 Canvas

```
        void NPACcanvas (OBJECT* object_name,
                                int x, int y,
                                int width, int height,
                                (void*) action_proc)
```

The object canvas places a graphical window on the screen at the given (x,y) location with the specified width and height. The action procedure specifies the output to the canvas. Normal procedures like moveto, drawline, drawpolygon, drawcircle are allowed. We do not go into details here because we do expect that other objects are of higher priority.

3.4.4 Graph

```
void NPACgraph (OBJECT* object_name,
                int x, int y,
                int width, int height,
                char* xlabel,
                char* ylabel,
                double xmin, xmax,
                double ymin, ymax,
                double* xvalues,
                double* yvalues,
                (void*) action_proc)
```

The object graph opens a two-dimensional graph plot on the screen with the obvious parameters. the x and y values are stored in the arrays `xvalues` and `yvalues`. The axis dimensions are specified with a minimal and maximal value.

3.4.5 Barchart

```
void NPACgraph (OBJECT* object_name,
                int x, int y,
                int width, int height,
                char** xlabel,
                char* ylabel,
                int xmin, xmax,
                double ymin, ymax,
                double* yvalues,
                (void*) action_proc)
```

The object barchart opens a two-dimensional graph plot on the screen with the obvious parameters. the x and y values are stored in the arrays `xvalues` and `yvalues`. The axis dimensions are specified with a minimal and maximal value. The x axis represents integer coordinates. Instead of Just one label for the x-axis we have for each xvalue a label.

3.4.6 Photo

```
void NPACphotoRead (OBJECT* object_name,
                int x, int y,
```

```
                                int width, int height,
 char* filename,
 char* path,
                                (void*) action_proc)


        void NPACphotoPlace (OBJECT* object_name,
                                int x, int y,
                                int width, int height,
                                PHOTO* picture,
                                (void*) action_proc)
```

The photo routines place a given photo specified in some format (bitmap) on the screen. We suggest to use the ppm format because it is supported by many packages and has a high compression rate. While the read routine reads a photo from a file the place routine places a photo stored in the local memory on the screen.
One very important routine will be to display a movie of a number of frames. This is supported by the movie routine.

```
        void NPACphotoMovie (OBJECT* object_name,
                                int x, int y,
                                int width, int height,
 char* dirpath,
                                (void*) action_proc)
```

Here the parameter `dirpath` points out a directory in which a number of frames are located. The frames are read in in alphabetical order and displayed with a given delay on the screen. Figure 2 shows the outline of a movie object. Two modes are distinguished. The animation mode supports the continuous play of the frames in the directory while the frame mode allows to go through the frames interactively. Switching between the modes is possible the `reset` button places the pointer to the active movie from to the first frame in the directory. Extensions like repeat might be useful too.

4 Future

This document should show that it is desirable to have a common interface between different applications. The library could be written with the help of existing graphical users packages. The once suitable for it would be AVS and TCL/tk. Other GUI builders might be worth while to consider. The author is in favor of TCL/tk because it is very fast and supports already most of the graphical objects needed by a simple application. A raytrace package is also available under the name YART. TCL/Tk is a public domain software and it is easy to install it on the different machines available at NPAC. The often mentioned complaint about AVS that it uses almost all the CPU time could not be found with TCL/Tk.

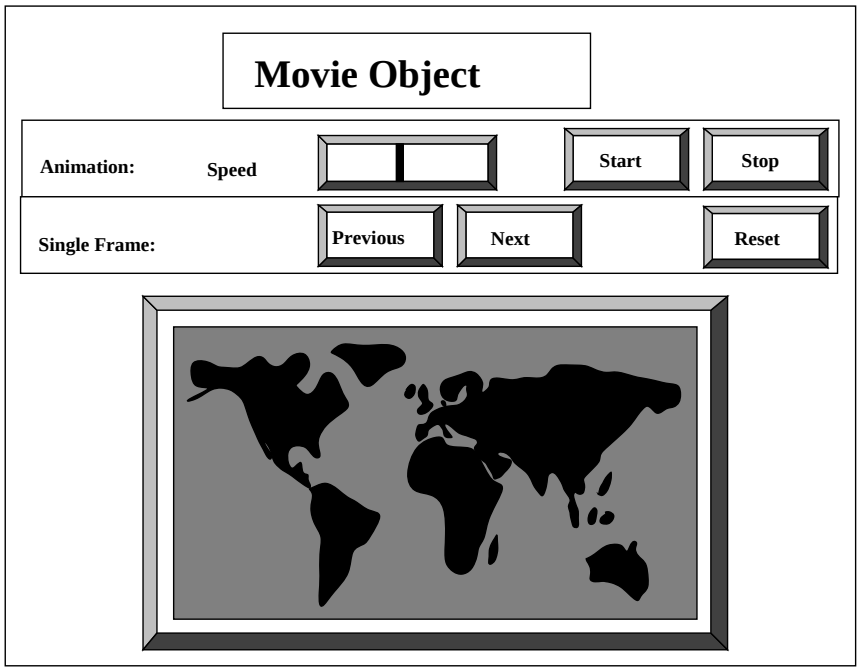

Figure 2: The essential graphic objects

If the one could generate simple example programs in AVS explaining the use of the graphical objects, than a applications programmer would be in the advantage to reduce the time for learning AVS. In contrast to using AVS the startup time for using the standard NPAC GUI library would almost be zero.

References

[1] Computer Science Dictionary. Gopher Service in UK.

[2] *AVS Users Manual*, 1993.

[3] Delmans, S. Design and Implementation of a Programming Environment for Intercactive Construction of Graphical User Interfaces. Tech. rep., Technical University Berlin, 1993.

[4] Ousterhout, J. K. *Tcl and Tk Toolkit.* Addison-Wesley, 1994.

[5] University Virginia. *SUIT – SImple User Interface Tool*, 1992.

A Dictionary

From [1] we find GUI pakages and some essential terms:

A.0.7 Terms

Graphical user interface (GUI) The use of pictures rather than just words to represent the input and output of a program. A program with a GUI runs under some windowing system (eg. X-Windows, Microsoft Windows, Acorn RISC OS). The program displays certain icons, buttons, dialogue boxes etc. in its window on the screen and the user controls it by moving a pointer on the screen (typically controlled by a mouse) and selecting certain objects by pressing buttons on the mouse while the pointer is pointing at them.

Widget In graphical user interfaces, a combination of a graphic symbol and some program code to perform a specific function. Eg. a scroll-bar or button. Windowing systems usually provide widget libraries containing commonly used widgets drawn in a certain style and with consistent behavior.

widget n. 1. A meta-thing. Used to stand for a real object in didactic examples (especially database tutorials). Legend has it that the original widgets were holders for buggy whips. "But suppose the parts list for a widget has 52 entries...." 2. [poss. evoking 'window gadget'] A user interface object in X graphical user interfaces.

A.0.8 GUI Packages

DataViews Graphical user interface development software from V.I.Corporation, aimed at constructing platform-independent interactive views of dynamic data.

Fresco An object-oriented API for graphical user interfaces, under development by the X consortium as an open, multi-vendor standard.

GUIDE Graphical User Interface Development Environment from Sun.

InterViews An object-oriented toolkit developed at Stanford University for building graphical user interfaces. It is implemented in C++ and provides a library of objects and a set of protocols for composing them.

Motif The standard Graphical User Interface and window manager from OSF, running on the X Window System

NewWave A graphical user interface and object-oriented environment from Hewlett-Packard, based on Windows 3.0 and available on UNIX workstations.

Open Look A graphical user interface and window manager from Sun and AT&.T.

UAN User Action Notation. VPI. A notation for representation of graphical user interfaces, e.g. mice and icons, H. Hartson et al, ACM Trans on Info Sys, July 1990.