An introduction to InterViews

Jean-Claude Dulac, Dave Nichols, Jos van Trier

ABSTRACT

Several members of SEP have started writing interactive programs using a software package called InterViews. We have chosen to do this because InterViews contains a very powerful object-oriented model of user interface, and it is written on top of standard environments: C++, the X-Window system, and UNIX.

INTRODUCTION

InterViews is a user interface package written in C++. It is described by the authors (Linton et al., 1988) as a package that "supports the design and implementation of user interfaces." It is is currently implemented as a layer of software on top of the X-Window system. This means that it will be portable to many different computer systems ranging from personal computers, through graphics workstations to supercomputers. Any system that support Unix, C++ and X-Window should be able to run InterViews. X-Window is the most widely available windowing system and has been implemented on almost all unix systems. It has been made freely available by MIT and is therefore widely used. The latest version of InterViews will be distributed on the next X-Window release tape from M.I.T.

Although C++ (Stroustrup, 1986) may not be the first choice of many objective language purists it has gained wide acceptance. A future version of the Unix operating system to be written by AT&T and Sun Microsystems will be written in C++. The use of an objective language means that we are able to extend the InterViews functions with minimum effort. As described by Rick Ottolini in a previous report (Ottolini, 1987) objective languages help you write code that is modular and reusable.

InterViews has been developed by members of the CIS (Center for Integrated Systems) department at Stanford University. Because it is locally developed we are

able to consult with the designers, obtain expert advice and give them feedback about features we wish to see implemented in future versions.

Though none of us had used C++ before starting to use InterViews we found that the development time for new interactive applications has been much less than we would have expected using other languages and window systems. Now that we have some experiences using InterViews and C++ we are confident that any further applications can be implemented even faster.

A BRIEF DESCRIPTION OF INTERVIEWS

InterViews (Interactive View) is a library of classes that can be used to construct a graphical user interface from interactive components. Only few primitive classes make direct calls to X-library functions. Neither the higher level classes nor our applications contain any X-calls.

Interactors

In InterViews each interactive component, called an *Interactor*, has a preferred shape and size. The preferred shape and size of a compositions of components, a *Scene*, is calculated from those of the components. The Interactor is responsible for making the best use of the display space actually allocated to it. InterViews separates the output style and the interactive style of the Interactor. Then, the interactive objects can be easely derived to suit both, the output behavior and the interactive behavior, of a particular application.

All interactors support the following set of operations that characterize their behavior. These operations may be applied to all objects derived from the Interactor class:

- *Draw:* The draw operation defines the appearance of the interactor. Calling Draw causes the interactor to display a representation of itself on its canvas. If the interactor is a scene, it will also call Draw on each of its components.
- *Redraw:* Redraw all or part of the interactor.
- *Update:* Update an interactor, something may have been changed.
- *Resize:* The interactor's canvas has changed size.
- *Handle:* Specify its behavior with respect to an input event.

An Interactor interested in input events has a *Sensor* that defines its current input interest (e.g. mouse motions, mouse clicks, keyboard events). If an interactor is not interested in a particular type of event, the Handle function for that interactor will not be called. Each event is targeted to a particular interactor. An interactor can receive input events in two ways. It can read the next event from the (global)

input queue, or an event can be passed from another interactor using the Handle function. A purely event driven organization can be produced by using the following C++ code fragment:

```
Event e ;
for(;;;) {
     Read(e) ;
     e.target-> Handle(e) ;
}
```

The meaning of this code is, in an infinite loop, read an event and call the event handling function of the interactor that is the target of the event.

Each Interactor internally maintains a record of its *Perspective* that defines the users view of the object, what is visible, whether it is zoomed etc. This means that all classes derived from interactor can be zoomed, and scrolled without the programmer having to write any new code describing how to handle zooming and scrolling.

Scenes

The base class for composite objects is the *Scene.* Scenes define operations for managing their components:

- *Insert and Remove:* basic operations to add and subtract components from a scene.
- *Raise and Lower:* change the front-to-back ordering of components into a scene.
- *Move:* notify a change in the position of a component within a scene.
- *Change:* notify a change in the shape of a component within a scene.
- *Propagate:* propagate changes to component interactors.

Structured Graphics

The *Graphic* class is an example of a higher level class in InterViews. Almost all programs that produce a picture within a window do so using the Graphic class and its derived classes. Each Graphic object has its own graphic state that includes attributes such as color, line style, and coordinate transformations. All Graphics can draw and erase themselves and provide operations for examining and changing attributes, performing hit detection, moving, scaling and rotating themselves.

Subclasses of Graphic include *Line*, *Circle*, *Rectangle* and most importantly *Picture*. Pictures are the basic mechanism for building hierarchies of Graphics. It is a Graphic that contains a collection of other Graphics. A Picture draws itself by drawing each component with a graphic state formed by combining the component's state with its own. So if you scale a picture all the Graphics within it will be scaled.

Other high level classes

InterViews also provides classes that support the following interactive objects amongst others.

- Buttons, of three types (Push buttons, Check boxes, Radio buttons)
- Menus
- Scrollbars and Panners, used to change the users view of an interactor.
- Structured Text classes for intelligently displaying text.
- Boxes, Trays and Viewports for arranging other objects on the screen.
- Rubberband objects that can track the cursor across the screen.

Every program using InterViews must create a *World* object. This object represents the root scene of a display, when initialized it opens a connection to the window server. As with all X-Window's displays the window server does not have to be running on the same machine as the program generating the display requests.

ADVANTAGES OF OBJECTIVE PROGRAMMING

The main property of objective languages that promotes code reuse is "inheritance" this means that a new class derived from an existing class inherits all the properties of the class and only new or different properties need to be defined. The great advantage of InterViews is that all the basic classes required for creating interactive programs have already been written by someone else.

As an example consider the task of writing a class that plots a "vplot" file (vplot is SEP's internal graphics language) on the screen. We can derive a class from *Picture*. All we need to add is one variable that defines the vplot file to be read and one new function for this class that interprets the file and creates *Graphic* objects that it adds to itself. If we create an object in this new class and call the read file function, we can thereafter treat it as a Picture. It can be drawn, moved, scaled, rotated, since these operations are defined for the parent class, Picture.

The most difficult part of writing an objective program is deciding on the basic classes on that all our programs will be based. By using InterViews we have bypassed this step, somebody else has done most of the hard work for us.

Example

The following code fragment shows how a Graphic object is moved on the screen:

```
void Movetool::Move(Event& e) {
    Coord x0, y0, x1, y1, x2,y2 ;

    Graphic* target = Select(e);                                    (1)
    target->GetBox(x0,y0,x1,y1);                                    (2)

    SlidingRect slidingRect(nil,nil,x0,y0,x1,y1,e.x,e.y);           (3)

    do {                                                            (4)
        Read(e);
        slidingRect.Track(e.x, e.y);
    } while( e.eventType != UpEvent )

    slidingRect.GetCurrent(x2,y2,x1,y1);                            (5)
    view->Move(target, x2-x0, y2-y0);                               (6)
}
```

The English description of this code is :

1. We are passed an event "e" , "target" is the object selected by that event.
2. Find the bounding box of "target".
3. Make a new sliding rectangle whose initial coordinates are the bounding box.
4. Read events and make the sliding rectangle track the cursor. If the event is a mouse key being released, UpEvent, stop tracking.
5. Get final coordinates of the sliding rectangle.
6. Move the object "target" by the the difference between the original and final coordinates of the left bottom sliding rectangle.

The importance of using objective programming is that this code fragment will work correctly for any Graphic object, or any object of a class derived from Graphic. The object "target" selected by the cursor could be a circle, square, a collection of graphics (Picture), or the "vplot" object described earlier. Because they are all derived from Graphic they can all be treated as a Graphic. The extra functionality provided by InterViews means that this code will work correctly even when the users view of the data has been scaled, rotated, or scrolled.

CURRENT DRAWBACKS

There are two main drawbacks to using InterViews at the moment. The first is that there is not a *Graphic* class that supports byte rasters. This means that at the moment we are unable to display grey level seismic data as easily as we display other graphic objects. The InterViews designers have responded positively to our request for this facility and we look forward to seeing byte rasters incorporated into a future version of InterViews. In the meantime we will write our own class supporting byte rasters so that we can proceed with interactive program development.

The other drawback is the speed of the graphics rendering on our Sun workstations. The programs run at a satisfactory speed when using release 10 of X-Window but they run far too slow to be practically useful when using release 11 of X-Window. The InterViews designers are using release 11 for all their development work so we will have to move to version 11 soon. The X-Window software we use at the moment is the MIT code that is not optimized for particular machines. Sun Microsystems is soon to release a version of X-Window version-11 that is specifically designed for Sun workstations. This product has been much delayed, it was due for release earlier this year but is now due for release in November. We hope that our speed problems will disappear when this new version is released.

CONCLUSIONS

Use of InterViews relieves the programmer of much of the work in writing an interactive program. The classes available in InterViews are well designed and easily extendable to deal with specific problems. The use of an objective language, C++, makes for code that is portable between different programs. We already use many of each other's program modules in our interactive programs. We think it would not have been possible to write our applications in such a short time and with so little programming effort if we had used SunView and C. The combination of X-Window and C++ means that the programs should be easily ported to other machines and our work will not become obsolete if new hardware is purchased.

REFERENCES

Linton M., & P. Calder, 1987, The design and implementation of InterViews: *in* Proceedings of the USENIX C++ Workshop, Santa Fe, NM, 256-267.

Ottolini, R., 1987, Techniques for organizing interactive graphics programs; SEP–**51**, 421-427.

Stroustrup, B., 1986, The C++ programming language; Addison-Wesley