An Object-Oriented Approach for Supporting Both X and Terminal Interfaces

J. Johnson (STScI)

While the X Window System is clearly becoming the dominant user interface display system, there is still a continuing need to support character cell terminal devices. For a user interface application which must support both, it is desirable to be able to use them to their fullest extent, as opposed to simply providing a character-based display within an X window. An object-oriented application framework is presented here which allows the full capabilities of each system to be used while minimizing and isolating the amount of device-dependent code.

Every user interface application consists of two parts: the various components used to display information and accept user input, and the processing of the interaction between these components. Many user interfaces are built around a core set of components such as menus, text entry fields, and forms. For a given application, the interaction between these components is the same regardless of the display system in use.

Our approach is to implement each component as an object, accessible via a single public interface. All of the code necessary to implement the component for the desired display system is completely encapsulated within the object. The application is then written as a collection of objects interacting in a device-independent manner with one another. If a display system provides additional capabilities, the application can be extended by adding objects (e.g. for image display in X).

This approach provides several benefits. First, the application can be ported to any display system capable of implementing the various components. Of course, the closer the system matches the set of components, the less code will be necessary to implement each object. To implement a text entry field using OSF/Motif is fairly trivial since a TextField widget is already defined, whereas implementing the same object in Curses would require considerably more effort. Second, the objects are application-level components, not low-level as provided by the display system. This allows arbitrarily complex components to be developed using both the display system library (e.g. OSF/Motif) and other application objects. For example, a graphical skymap object could be implemented as a form object containing pushbutton and text entry field objects, as well as calls to the display library to perform the various drawing operations. Third, other user interface applications can be developed using the same set of objects, promoting software reuse.

We are currently implementing StarView, the user interface to the HST science archive (ST DADS) using this approach. StarView will be written in C++, and will use Vermont Views for the terminal interface and OSF/Motif for the X Window interface.