Public Access Programming: Opening The Black Box That Hides Internal Data

E. Mandel

Harvard-Smithsonian Center for Astrophysics, Cambridge, MA 02138

R. Swick

Digital Equipment Corporation, Nashua, NH 03062

Abstract.

We describe a simple and effective use of the *X* Toolkit (*Xt*) selection mechanism by which data in an *X* program can be tagged with string identifiers and then accessed externally by other programs. We will discuss our design goals, the technical challenges we faced – including extensions to the *Xt* selection implementation – and the user interface and application programming interface that we developed to meet these challenges. We also will discuss the important implications that our scheme has for programs such as *SAOimage*.

1. Introduction

Data analysis and display programs typically are written as "black boxes" with only a limited specification of input and output data. Such programs are linear and transient: they are activated in response to a single event (such as a "run" command) and they exit after transforming their input data into output data. Because these programs do not persist in memory after having completed their task, their internal data also do not persist and are not available to external processes.

There are analysis programs that *do* persist over time and are not linear in their operation. Rather than performing a single task once, they perform an action (or different actions) repeatedly in response to user events. The internal data of these event-driven programs generally are not available to external processes either, but unlike the transient programs mentioned above, there is no compelling reason why this should be so. Indeed, there are important cases where it is desirable to read or even modify a program's "internal" data through external means. The *SAOimage* image display program is a good example: over the years there have been numerous requests by users and developers to be able to read and/or write *SAOimage*'s image data, region information, colormaps, *etc.*, externally.

It even can be argued that external access to "internal" data is critical for the proper integration of programs such as *SAOimage* into the heterogeneous data analysis environments used by astronomers today. A good image display program should be able to interface with the many analysis tools now available.

To do this, it must be able to send data and information to analysis programs for further transformation, and also be able to receive the results of these transformations for further display.

These considerations have guided efforts at Smithsonian Astrophysical Observatory to develop a successor to the popular *SAOimage* image display program. From the beginning, we have recognized that the central problem we face is not the display of data, but how the image display program will interact with other programs and systems. Indeed, our discussions with users and developers have centered on the need for the image display to be capable of being controlled by external processes (which send it data and information) and also to be capable of controlling processes (by sending data and information to those processes). For example, many users have requested that *SAOimage* support quick-look analysis of the displayed data, *i.e.,* that *SAOimage* be able to launch arbitrary external processes to retrieve the image data, transform it, and display results, perhaps even by writing data back to the image display. This kind of functionality will help make maximal use of existing analysis software.

2. XPA Design Goals

We therefore set about to develop a general mechanism by which arbitrary programs could communicate data and information to and from *X* programs such as *SAOimage*. From the outset, our design goals included the following:

- The mechanism should be driven by the external process, which either sends "named" data or requests "named" data to or from an *X* program.
- The mechanism should allow for the simultaneous sending and receiving of data associated with the same "name".
- *X* programs should send or receive data using the standard event-driven callback paradigm already familiar to *X* application programmers.
- External processes should be able to access data in *X* programs using a simple application programming interface (*API*) that does not require knowledge of *X* or linking against the *X* libraries.
- In addition to a low-level *API*, there should be a set of high-level programs that send or receive data at the command line.
- The mechanism should not add substantial overhead to the application, so that it may be liberally applied and remain dormant until invoked.

3. Layered Implementation of XPA

The result of our efforts is the "*X* Public Access" mechanism (*XPA*), a layered interface using *X* Toolkit (*Xt*) selections, by which data in an *X* program can be tagged with string identifiers and then accessed externally by other programs. We chose to implement *XPA* using the *Xt* selection mechanism because of the

universal availability of the *Xt* Toolkit and also because the selection mechanism hides the platform-dependent implementation of the interprocess communications. Layering *XPA* on *Xt* selections also allows an analysis program to be executed on a remote network host with no additional effort on the part of either the programmer or the user.

The *XPA* interface consists of three main parts:

- The *NewXPA* subroutine is used by *X* programs to tag data with a string identifier and to register send and receive callbacks for these data.
- The *OpenXPA* subroutine is used by external applications to exchange data with an *X* application.
- Two high-level programs, *xpaset* and *xpaget*, allow data to be sent or received from the command line.

The *XPA* mechanism is conceptually rather simple. An *X* application such as *SAOimage* calls the *NewXPA* routine to associate a string name with a particular piece of data. The routine allows one to specify send and receive callback procedures which will be executed by the program when an external process either asks for this data or sends new data. Either of the callbacks can be omitted, so that a particular data item can be specified as being read-only, read-write, or write-only. Having tagged one or more data items in this way, the *X* program creates its graphical interface and enters its usual *X* event loop.

The tagged data are retrieved using either *xpaget* (at the command line) or *OpenXPA* for reading (inside a program). Both take the name of the tag as input. The *xpaget* program returns the data from the *X* application through its standard output. The *OpenXPA* routine (called for reading) returns a Unix file handle which can be used to retrieve the data using standard I/O calls. The *xpaset* program will send data from its standard input to an *X* application, while *OpenXPA* (called for writing) will return a Unix file handle through which data can be written to the *X* application using standard I/O calls. Note that *OpenXPA* simply opens a pipe to *xpaset* or *xpaget*. The *XPA* code is not added directly to the application and thus the *X* libraries need not be linked into the external program.

Arbitrarily large amounts of data can be transferred to and from *X* programs using *XPA* (subject, of course, to limitations on available memory). The system is designed so that the data associated with a given tag can be read and written simultaneously. This is accomplished by having the application manage its own "current" copy of the data, which data are sent to external requesting processes. Newly received data are maintained internally by *XPA* until the user-specified receive callback can be executed safely to replace the old data with the new. Because of this double buffering, a program can retrieve data from *SAOimage*, transform it in some way, and then send the transformed data back to *SAOimage* for display with one filter command, such as: *xpaget* data | transform | *xpaset* data.

Other features of the *XPA* interface include the ability to retrieve the list of tagged data items in an application and the ability of an external program to include an arbitrary "parameter" string in communications with the *X* application, so that the latter can use this information when transferring data.

4. Extensions to the Xt Selection Mechanism

The *Xt* selection mechanism, which allows *X* applications to exchange data, required extensions in order to implement *XPA*. The selection mechanism, as supplied by MIT, relies on timeouts to determine whether a transfer has been interrupted. For *XPA*, we had to disable the *Xt* timeouts in order to allow the computation and sending of data over an arbitrarily long period of time. This would be the case, for example, if a convolution program was sending image data to an *X* application one line at a time.

Instead of timeouts, therefore, *XPA* uses two mechanisms to ensure that both sides of a transfer are active. The external receiver code (for *xpaget* or *OpenXPA*) sets a special *X* property on an *X* window. The sending process monitors this property. If the receiver terminates, the property will disappear and the sender can take appropriate action to abort the transfer and clean up. On the other hand, to monitor a sending process, the receiver code checks to make sure that the owner of the selection is the same window as the original owner when the receive request was issued. If the selection owner changes, this means that the original sender is no longer sending data. The receiver then can take appropriate action.

5. Use of XPA in SAOimage

As previously mentioned, the *XPA* interface will be used by *SAOimage* to allow any external program to retrieve information or to control the display directly, by accessing the tagged data items in the program. Programs will be able to load images, modify the colormap scheme, get or set regions of interest, *etc.* Note that *SAOimage* itself will use the *XPA* functionality to "externalize" algorithms that usually are considered to be "internal". One very important application of the externalization of data transformation algorithms is the use of separate file formatting programs to convert different data formats into a standard *FITS* format that will be understood by *SAOimage*. Under this scheme, new file formats will be supported simply by writing a *FITS* converter that interprets the data and sends it directly to *SAOimage*, without having to change the *SAOimage* program itself. This will allow different projects to display their own private file formats easily.

6. Conclusion

XPA is a simple interface that allows any program to communicate arbitrary data to and from an *X* program. By facilitating the sharing of data between programs, *XPA* takes us one step closer to an era in which our different analysis tools and systems can work together as an integrated whole.

Acknowledgments. This work was supported under NASA contracts to the *IRAF* Technical Working Group (NAGW-1921), the *AXAF* High Resolution Camera (NAS8-38248), the *AXAF* Support Team (NAS8-36123) and the *AXAF* Science Center (NAS8-39073).