Astronomical Data Analysis Software and Systems VI
ASP Conference Series, Vol. 125, 1997
Gareth Hunt and H. E. Payne, eds.

A Java Interface For SkyView

Thomas A. McGlynn,[1] Keith A. Scollick,[2] and Nicholas E. White

NASA/Goddard Space Flight Center, Greenbelt, MD 20771,
E-mail: tam@silk.gsfc.nasa.gov

Abstract. We discuss our experiences in building a Java interface to the *SkyView* virtual observatory. We describe the reasons we choose Java, the problems we encountered, and how we feel this language may be used in the future for Web and stand-alone applications.

1. Introduction

The advent of Java applets has provided a new means by which archive systems can serve users. This paper discusses the reasons for, and experience gained in building a Java interface to the *SkyView* facility developed at the High Energy Astrophysics Science Archive Research Center (HEASARC). We address the capabilities we hoped to achieve in the applet, how the applet paradigm matches with our system, and the strengths and weaknesses of the extent Java implementation. We conclude with a discussion of the success of our preliminary interface, and how we would assess the suitability of Java for other applications.

2. Background

SkyView was designed as a virtual telescope on the Web. The basic concept is that a user specifies a position or object in the Sky and a wavelength regime (or survey), and *SkyView* returns a image with any desired geometry: coordinate system, scale, orientation, projection, ...

Prior to our Java development there were three mechanisms for accessing *SkyView*. By far the most popular is through a set of forms on the World Wide Web (WWW). Three forms of varying complexity allow the user to specify the image desired and return the image as a dynamically generated Web page. There is also a batch interface, where users who have installed some minimal software (a pair of Perl scripts) on their machines may make a request for a *SkyView* image from the command line of their local computer.

These interfaces address that fraction of *SkyView* use where the user understands exactly what image is needed for further analysis. However, a third interface is provided to allow users substantial image manipulation capabilities. *SkyView* can generate images with a large amount of information. *SkyView*'s Interactive Interface allows users to play interactively with the color table, catalog

[1]Universities Space Research Association

[2]Computer Sciences Corporation

overlays, contours, coordinate grids, and other overlays to help understand this information. This interface is implemented as a remote X-windows task started on the *SkyView* server. Because of this implementation, the interface can be extremely slow, does not support Macintosh or PC hosts, and requires users to adjust the security environment of their machines. These problems substantially compromise the usefulness of the interface.

3. Why Java?

Java allows us to re-implement our Interactive Interface in a way which addresses these concerns. Since image manipulation occurs locally, it can be substantially faster than when each image update needs to be transmitted over the Web. Popular Web browsers for all major hardware types support Java. While the security issues are not entirely resolved, they are placed in a context where they can be addressed simultaneously for *SkyView* and other interactive systems.

Beyond these goals, we have hoped that Java will also address concerns we have with our current Web forms. While the idea of filling out a form and then submitting a request is essentially the *modus vivendi* the Web, there are substantial shortcomings in the simple HTML implementations of forms. For example, it is virtually impossible to create a clean-looking form that supports a sophisticated set of options. Since Java supports a full GUI interface for applets, it becomes possible for us to offer a single interface which is simultaneously powerful and easy to use.

Similarly, the HTML form interfaces have less than ideal mechanisms for documentation and error reporting. A user must move back and forth between form and documentation pages. Users must submit a request and get an error document back when an error is made, even when the error is egregious. In a full GUI, we can use pop-up error windows and interface hiding to ensure that users' navigation of *SkyView* is as smooth as possible.

4. Putting SkyView in Java

SkyView's architecture has facilitated a relatively straightforward integration of a Java interface. All of the *SkyView* interfaces can be broken into three elements: the geometry engine which accesses the archive and actually creates images, an image annotater which provides overlays on images and other services such as transforming image formats, and the user interface. The various user interfaces call first the geometry engine, then the image annotater, and finally display the images. Since these elements are already decoupled it is easy to add Java as another user interface.

Our current Java interface incorporates many features that would be difficult or impossible using a purely form driven system including:

- simultaneous retrieval of multiple images,
- blinking of images,
- hiding many complex options within a menu bar,
- color table manipulation,

- displaying the coordinates and value of pixels using the mouse, and
- "greying"-out options that are not currently usable.

Development of additional capabilities is proceeding rapidly. The *SkyView* Java interface is available from the *SkyView* home page.[3]

5. Assessment of Java.

Overall, our experience with Java has been reasonably good. Usable interfaces were built with only a few weeks of work. The language is clear and understandable, and seems to be an excellent *entreé* to the object-oriented world.

The difficulties we encountered were not those we anticipated. The most intractable problems have come from relatively simple tasks, such as laying out the GUI in pleasing fashion, or doing such basic tasks as formatted I/O. These have pointed out annoying, and occasionally alarming, deficiencies in the Java Applications Programmer Interface (API).

Some failings reflect the languages immaturity but, regardless, they make it difficult to write real code. Java does not appear to have made support for scientific programming an important element of its design. There is no support for formatted I/O or array I/O . In general, I/O is handled quite clumsily. The lack of support for operator overloading obfuscates scientific code, where the use of multiplication and addition for non-scalar quantities is well founded. Nor is there any notion of support for simple array operations.

The Abstract Windowing Toolkit (AWT) provides reasonable functionality for individual components. However, it can be extremely difficult to lay out a page in the fashion desired. The `GridBagLayout` class, which is intended to address this issue, is confusing and poorly documented.

Another handicap of programming in Java is the lack of existing libraries. In other languages, this can be addressed, in the interim, by calling existing libraries in other languages. However, Java applet programming does not permit linking to foreign language modules. For *SkyView*, the lack of FITS and geometry libraries is particularly problematic. If Java is to be popular within the astronomical community, a FITS library is essential.

The Java security model is extremely frustrating. While we understand the rationale for the high level of security required for Java applets, the lack of any ability to read from or write to the user file system (in a portable fashion) is extremely annoying. It markedly reduces the functionality we are able to offer in Java and reduces the efficiency of the functions we can offer, e.g., an image must be loaded to the Java interface and then separately downloaded to the user when needed.

6. Future Plans

While we have encountered a number of frustrations in the Java interface, it can clearly meet our original goal of being an effective replacement for our Interactive Interface. Our next step is to continue development, so that it provides

[3]http://skyview.gsfc.nasa.gov

a reasonable alternative to our forms-based interfaces. The ability to hide unneeded capabilities and disable features for which the user has not yet specified prerequisites, the capabilities for immediate feedback that helps users to correct errors in fields, and the possibilities for a really effective context-sensitive help, are all extremely attractive. We do not anticipate ever removing the forms interfaces, but we do feel that Java may become the primary *SkyView* interface as Java becomes more popular and efficient.

Other projects at the HEASARC are also beginning to use Java applets. We have already developed a preliminary applet for plotting the results of database queries, and we are looking at Java to provide an interface which can unify the various elements of our W3Browse database retrieval system.

Our conclusion is that Java can provide the capability to provide users with interactive access to our systems, but it is still very immature. In general, we see most of these applications following the same lines as *SkyView*. The applet provides the user interface to initiating services at our host site. The bulk of the CPU processing and any capabilities requiring large amounts of I/O are done at our site. Results are then returned to the Java interface, which provides capabilities to display and manipulate them locally.

Beyond Java applets, we have also begun exploring the use of Java for full applications. Here a number of the limits to Java (local I/O and linking to non-Java modules) are lifted. Java's strengths in portability, networking, and Web access make it attractive in a number of areas.