Astronomical Data Analysis Software and Systems X
ASP Conference Series, Vol. 238, 2001
F. R. Harnden Jr., F. A. Primini, and H. E. Payne, eds.

The PyRAF Graphics System

M. D. De La Peña, R. L. White, and P. Greenfield

Space Telescope Science Institute, Baltimore, MD 21218

Abstract. This paper describes the features, plans, and design for PyRAF graphics. PyRAF is an alternative CL for IRAF based on Python. Since IRAF tasks depend on the CL to manage all graphics, any CL replacement must implement a means of handling IRAF graphics.

We have developed graphics kernels for PyRAF written completely in Python that are capable of working with IRAF graphics tasks (including interactive tasks). We have added capabilities such as multiple graphics windows, a scrollable message input/output region, ability to recall previous plots and "undo" features, and automatic focus handling.

The design of the PyRAF graphics system makes use of Python's object-oriented features. We describe the design employed to isolate the details of IRAF's underlying graphics system from PyRAF, making it easier to support multiple kernels, including the ability to use the IRAF Graphics kernel tasks.

1. PyRAF Graphics Features

PyRAF is a new CL for IRAF (Greenfield & White 2000) that has been developed to allow writing scripts in Python that can run IRAF tasks and to allow enhancements to the interactive CL environment. The goal of the interactive environment was to retain the interface and syntax of the original IRAF CL to the maximum extent possible or sensible. Since IRAF expects the CL to handle all graphics, PyRAF must handle IRAF graphics.

The approach taken to the graphics system embodies a number of significant departures from that used by the IRAF CL for graphics. Like IRAF, we wish to retain the ability to use multiple graphics kernels. However, IRAF interactive graphics devices are largely terminal-based, whereas we decided to base our initial interactive kernel on a GUI library (through Python's Tkinter, though other multiplatform GUI libraries are available). Nevertheless, this does not preclude us from emulating the terminal-based graphics devices in the future. Basing graphics on a GUI library allowed us to add a number of features not easily added to the existing IRAF devices, such as dropdown menus, multiple graphics windows, and a scrollable status region.

Integrated with the menus is labeled access to all past plots in the current session, navigation through past plots via short-cuts (next, back, first, and last), the ability to create a new graphics window, plot edit features (undo, redo, refresh, and delete plot), ability to save the metacode to a file or print it (through existing IRAF kernels), and help for PyRAF graphics. A history

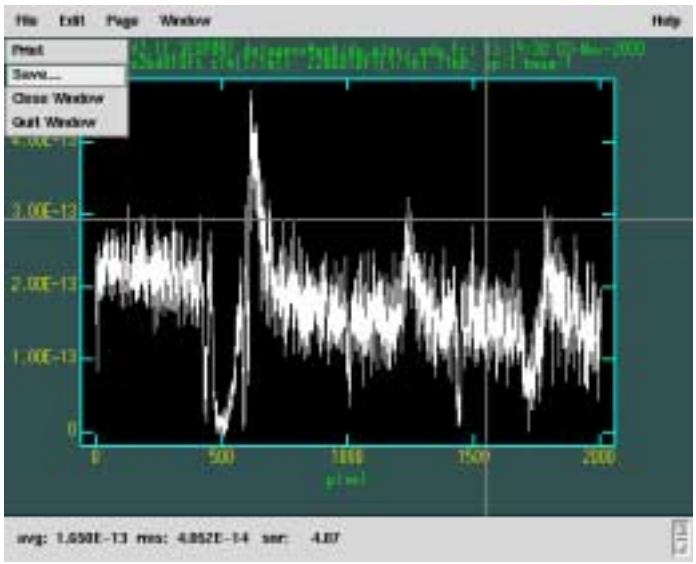

Figure 1. The basic PyRAF graphics window incorporates additional functionality accessible via a menu bar at the top and a buffer at the bottom of the window which contains a history of I/O messages.

buffer, implemented as a "status line," maintains a log of all input/output messages associated with the particular graphics window. The PyRAF system is capable of handling multiple graphics windows for a single PyRAF session. The graphics windows can be resized at any time, and window contents are subsequently resized accordingly. Most importantly for those who are longtime users of IRAF, PyRAF provides access to nearly all of the IRAF interactive features. The basic PyRAF graphics window is shown in Figure 1.

2. Planned Features

The initial release of PyRAF will contain most of the graphics features necessary for a viable system, but we envision further enhancements to the graphical environment. Some of the planned improvements for PyRAF are: alternative kernels to support graphics and image display, an enhanced "status line" buffer for easier history access, improved text rendering, optional balloon help (*aka* tool tips), a "preferences" menu (with options for color, printer choice, font size, etc.), an optional toolbar, additional IRAF interactive features ("capital" letter commands controlling roaming and zooming), use of GUIs for IRAF interactive commands versus special characters, keyboard accelerators, and the ability to plot Python data arrays to graphics windows (akin to IDL).

3. Open Issues

While we are attempting to retain most of the current IRAF CL's graphics features in PyRAF, there are some that have uncertain utility. We may decide

not to duplicate such features, particularly if there appears to be little demand for them. These include the ability to run IRAF kernels interactively, support for stdgraph devices such as xterm and xgterm, and some of the IRAF "capital" letter interactive commands.

4. Design Issues

One of the high-level goals of the PyRAF design was to eliminate any direct dependence on IRAF libraries (PyRAF only uses IRAF executables). The PyRAF interactive graphics kernel thus is written entirely in Python with no reuse of IRAF code. Since Python is interpreted, one issue was that of efficiency. This was alleviated in part by basing the graphics on OpenGL, yet most of the action takes place in Python. While graphics rendering is perceptibly slower on slower workstations (e.g., Sparc 4s), the speed is not objectionable. On newer machines, the difference is rarely noticeable.

An important objective of the PyRAF graphics design was to make it easy to support multiple interactive graphics kernels as well as non-interactive ones. While designing a kernel class that has a simple interface is straightforward, there are some tricky issues dealing with the fact that one graphics kernel may be asked to switch to another in midstream while retaining the appropriate metacode information and state for the new kernel to start. Multiple graphics windows introduce further complications (the solution was to instantiate a kernel for each window).

OpenGL provides powerful plotting capabilities (and a great deal that is not needed for simple 2-D plotting). Most of the IRAF metacode plotting primitives are quite easy to render in OpenGL. The only exception is text rendering. For simplicity and portability, the initial OpenGL graphics kernel relies on a simple stroked font implementation. Handling keyboard focus properly when multiple interactive windows are available (including an image display window) also presents special problems and required a few C routines to provide Xlib focus manipulation functionality typically absent from most GUI toolkits. Finally, a full-screen cursor is rendered in software and requires careful handling of when it is and is not enabled. Python exception handling is the key to robust management of the full-screen cursor and other graphics state information.

A Unified Modelling Language (UML) diagram depicting the high-level design of the PyRAF graphics system is shown in Figure 2. GkiKernel is the base class for all graphics kernel implementations and is used to provide a standard interface to the IRAF process which communicates with IRAF tasks. The first level subclasses are responsible for handling the interactive graphics, switching between graphics kernels, and invocation of the IRAF builtin kernels. Specifically, GkiInteractiveBase is the base class for *interactive* kernels and implements the supporting functionality (e.g., menu bar, status line message buffer, page caching, etc.). The specific interactive classes are GkiOpenGlKernel and GkiTkinterKernel. GkiOpenGlKernel is the OpenGL graphics kernel implementation which uses OpenGL (or Mesa) in combination with Tkinter (a Python version of Tk) to render the plots. Alternatively, GkiTkinterKernel is the Tkinter graphics kernel implementation which uses only Tkinter to render the plots. GkiProxy is a proxy base class which implements the GkiKernel interface and

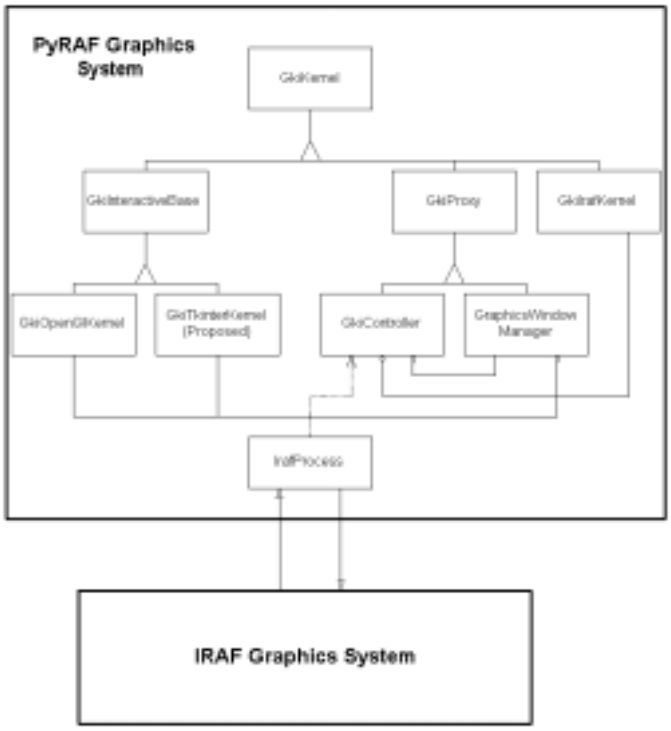

Figure 2. A UML diagram of the fundamental class relationships present in the PyRAF graphics system.

allows switching between different graphics kernels; GkiController is a GkiProxy which selects the active graphics kernel as directed by commands in the IRAF metacode stream. GraphicsWindowManager is a GkiProxy for the active graphics window which also acts as the manager for multiple graphics windows. GkiIrafKernel is a GkiKernel that routes metacode to an IRAF executable. Finally, IrafProcess handles the control and communication between the PyRAF and IRAF tasks and provides a unified interface to the IRAF subsystem.

5. PyRAF System Plans

A public beta version is currently available and the first release should be available by summer 2001. It is worth noting that PyRAF runs on IRAF-supported platforms without any changes to the IRAF system. For further details, users are encouraged to visit the PyRAF web site at http://pyraf.stsci.edu.

References

Greenfield, P. & White, R. L. 2000, in ASP Conf. Ser., Vol. 216, Astronomical Data Analysis Software and Systems IX, ed. N. Manset, C. Veillet, & D. Crabtree (San Francisco: ASP), 59