Astronomical Data Analysis Software and Systems VII
ASP Conference Series, Vol. 145, 1998
R. Albrecht, R. N. Hook and H. A. Bushouse, eds.

The New User Interface for the OVRO Millimeter Array

Steve Scott and Ray Finch

Owens Valley Radio Observatory, PO Box 986, Big Pine, CA 93513

Abstract.

A new user interface for the OVRO Millimeter Array is in the early phase of implementation. A rich interface has been developed that combines the use of color highlights, graphical representation of data, and audio. Java and Internet protocols are used to extend the interface across the World Wide Web. Compression is used to enable presentation of the interface over low bandwidth links.

1. Introduction

Modern astronomical instruments share the challenge of presenting a complex and changing system to observers, engineers and technicians. Caltech's six element millimeter wave aperture synthesis array at the Owens Valley Radio Observatory (OVRO) is in the process of implementing a new user interface to facilitate observing and troubleshooting of an increasingly complex instrument. Around the clock operation of the array is done by astronomers and students from a variety of institutions without the help of telescope operators. This style of operation works well when there is easy access to expert observers, instrumentation designers and maintenance personnel and when these people in turn have transparent access to the current state of the array. By porting the array interface to the Web, the instrument is available to everyone with an Internet capable system, independent of operating system or location. Extra steps have been taken to ensure that a modem connection provides a satisfying response for the user. Our current implementation has been done in the tradition of the Web, as we have released our software before completion. This premature release concentrates on the monitoring aspect of some of the most critical parts of our system and serves as proof of concept. The instant utility of this new capability indicates that this was a good choice.

2. Features

The design utilizes a client server architecture with a Java client providing the user interface on an arbitrary computer on the Internet. The Unix machine that runs the array is the host for the server programs which are written in C++. A user initially sees a menu[1] of 16 different monitoring windows that can be

[1] http:/www.ovro.caltech.edu/java/cma/menu.html

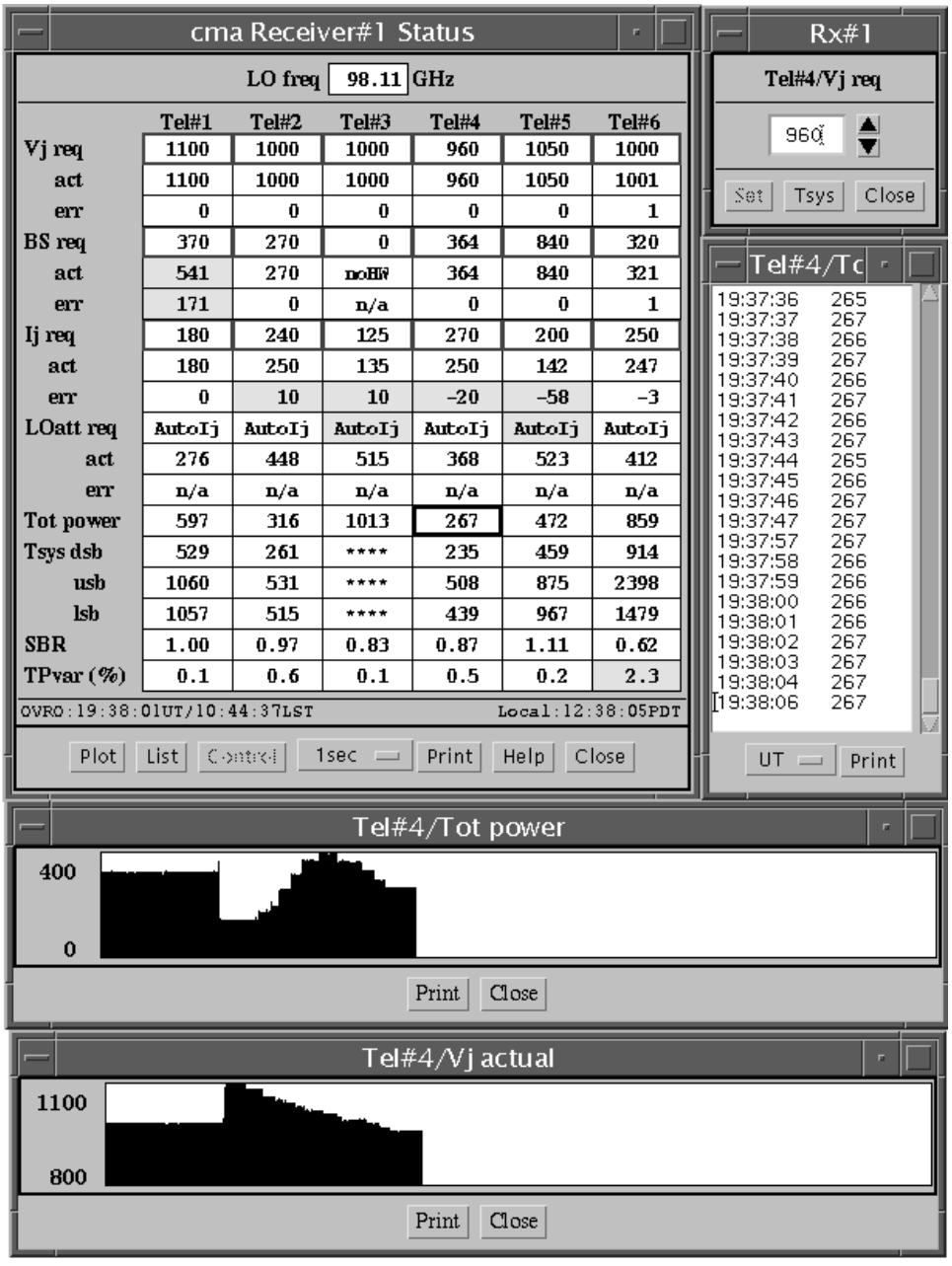

Figure 1. A single realtime window surrounded by its children.

launched onto their screen. Each of these windows is independent, and several of them are typically run simultaneously on the user's screen. The local window manager can be used to resize and place the windows to customize the layout as is illustrated in Figure 1.

The rectangles containing text on a light background are instances of the realtime data cell (RDC) that are the critical building blocks of the windows. The values within the cell are updated on a user selectable time scale that can be set from one second to thirty minutes using the choice menu at the bottom of the window. An RDC can optionally have the cell background color change as a function of the value in the cell. Typical color assignments are yellow for warning and red for alert. An audio chime can also be associated with the cell to emphasize the alert. When the window is initiated, each RDC begins to cache a history that can contain the most recent 200 samples. This history can then be accessed by other parts of the program. The RDCs are also used as part of a compact menu for selecting the advanced features. When an RDC is clicked with the mouse, its border is enhanced and it becomes the "selected" cell. In Figure 1 the "Tot Power" for telescope #4 is the selected cell. Clicking on the plot or list button at the bottom of the window brings up the plot or list window for that cell.

The plot and list features coupled with the history are an important part of the user interface for the RDCs and are shown in Figure 1. When a plot or list window is initiated it is seeded with the values in the RDC history. This allows the user to see an event occur in the main window and then to initiate a plot to look back in time before the event. Both plot and list contain print options. Additionally, the contents of the list window can be written to disk in a two column ASCII format. The plotting is intentionally simple. The ordinate is auto scaled but the abscissa simply uses one pixel per sample to allow a maximal number of samples to be plotted and avoid unnecessary rescaling. The ability of the list window to write to disk allows more complex analysis using a spread sheet or other tools. Because the histories for all of the RDCs within a window are synchronized, plots and listings are time aligned for cross comparison of cells. Each plot or listing contains an extension of the history buffer that can contain 1000 points.

Control functionality can also be added to RDCs. Using the same cell selection metaphor and the control button at the bottom of the window, a control widget can be launched. Cells with control capability are indicated by a purple outline rather than the standard black. An example of a control widget can be seen in the upper right hand corner of Figure 1. Security for control features is currently based on the Internet address of the client but will be changed to a login based mechanism using our Unix password files. If control is not authorized then the control button does not appear on the window.

Many of the RDCs are arranged in tabular form to provide a dense display and to emphasize the relationship of the data. It is also possible to lay out the RDCs in a freer format where there is a label associated with each RDC. These two styles can then be stacked vertically to provide a window with multiple sections.

3. Implementation

To eliminate the need for synchronized programming in the client and the server, the server programs send the client complete configuration information on startup and the client then configures itself accordingly. The result is clients

with significantly different appearances even though they are running copies of the same Java code.

The life cycle of a new window begins when the client opens a TCP/IP socket connection to a pre-assigned port on the array control computer with a request for a specific type of realtime window. The Unix system then runs a simple "Internet service" program attached to this port that forks a copy of the specific server program for the type of window requested. This establishes a one-to-one mapping between the client and a window specific server program. The client now requests its layout from the server, and after reacting to the reply it makes itself visible. It then begins a loop of requesting an update of the live data from the server, displaying the data, and then sleeping for the update interval. The multi-threaded nature of the Java environment allows reaction to user initiated events, such as plotting, while in this loop. Driving the update cycle from the client is appropriate for monitoring data as it is then very robust to unforeseen delays. These delays only cause a gradual degradation of response rather than total failure.

An efficient programming environment for the C++ servers is based on shared memory containing all of the data of interest about the array. This shared memory is fed by eight different realtime microcomputers embedded in the array hardware that send UDP datagrams twice a second. The server programs then simply map the shared memory into their address space to have access to the live data. The fundamental classes for the RDCs and layouts that are used to craft a window contain about 1500 lines of code. By using these classes a new server program can be written in a few pages, making the creation of a new window almost trivial.

4. Resources

Resource utilization is important in our choice of C++ for the server program because it is currently much more efficient than Java. Our midrange Unix processor can serve 50 realtime windows without substantial impact. At the Java client end, an average PC can easily display six realtime windows. The server to client data rate for three windows at a two second update is about 1KB/sec, which is a large enough fraction of the 2-3KB/sec available over a modem to make the response sluggish. To improve throughput, the data stream is compressed by transmitting only changes to the existing screen and escape codes that encode the number of unchanged characters to skip. This algorithm matches the data quite well when the screen appearance often shows just the twinkling of the least significant digits. Typical compression rates of 5 to 10 allow four or more windows to be displayed over a modem link with good response time.