The user interfaces for the NRAO-Green Bank Telescope

Ronald J. Maddalena[*]
National Radio Astronomy Observatory, P.O. Box 2, Green Bank, WV USA

ABSTRACT

The NRAO-Green Bank Telescope, the world's largest, fully steerable radio telescope, is now in routine use by visiting astronomers. The telescope is unique because of its offset optical design and its complex suite of state-of-the-art instruments. To exploit the full powers of the system, observers and staff members require intuitive user interfaces. We will demonstrate one of the various graphical user interfaces that have been built for the telescope. The interface, written in Tcl/Tk and used predominately by staff engineers, telescope operators, and programmers, is designed for very detailed monitoring and debugging of telescope components. The interface lies on top of an object-oriented control system that provides the GUI builder a set of software methods that is the same for every GBT device, from receivers to detectors. The uniform set of methods reduces the work normally needed in creating a high-level user interface and allows for the creation of interfaces in a range of programming languages.

Keywords: Radio telescope, control software, Tcl/Tk, user interfaces

INTRODUCTION

The Robert C. Byrd Green Bank Telescope (GBT)[1,2,3] is the world's largest, fully steerable, single-dish radio telescope with a projected aperture of 100 m. Since its dedication in August 2000, the telescope has been undergoing commissioning as well as hosting routine astronomical observing. The uniqueness and versatility of the GBT arises not only from its offset optics, plainly evident in Figure 1, but also from many of the other components of the telescope. This includes an active surface[4], a large suite of state-of-the-art receivers and detectors or backends, and a laser metrology system for real-time measurement of the pointing of the telescope and the shape of its reflector.

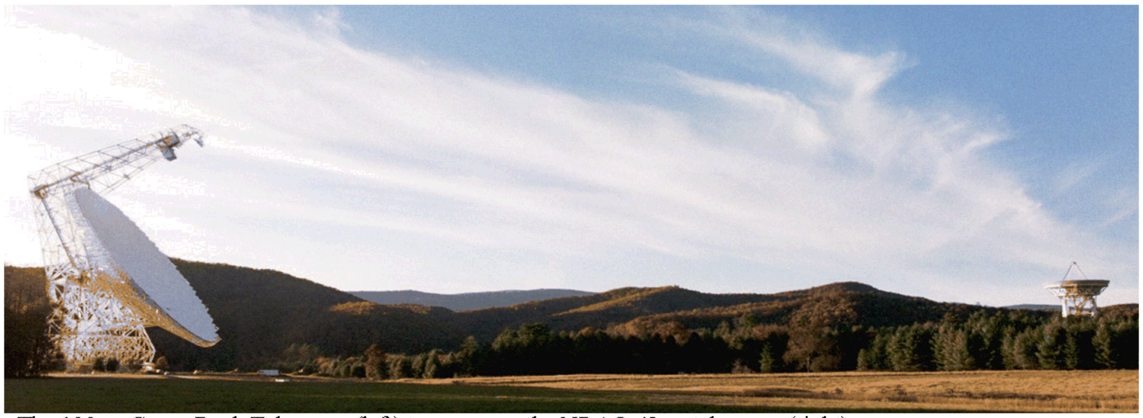


Figure 1: The 100-m Green Bank Telescope (left) towers over the NRAO 43-m telescope (right).

Almost every hardware component of the telescope is controlled and monitored almost exclusively through digital interfaces and their accompanying software. This heterogeneous set of telescope components requires a versatile and powerful software system that ties everything together to form a single, coordinated system. For almost every piece of hardware, this software system, called "Ygor", must be able to control all hardware components, provide monitoring

[*] rmaddale@nrao.edu; phone 304 456-2207; fax 304-456-2170; http://www.gb.nrao.edu

feedback, and handle messages and alarms. On top of the control system must be various user interfaces that provide levels of control or monitoring suitable for their intended users. Section 2 provides a brief overview of the "Ygor" monitor and control system and, in particular, the well-designed software communication interface between "Ygor" and user interfaces.

Section 3 describes the graphical user interface designed for support staff, engineers, programmers, and telescope operators. The interface, called "The Control Library for Engineers and Operators" (CLEO), was written in Tcl/Tk by one part-time staff member and two part-time undergraduates. Even though CLEO provides sophisticated controls and monitoring points for almost every aspect of the telescope, development was essentially on schedule with extremely low cost. Maintenance and on-going development amount to less than 5% of a staff member's time and user feedback has been extremely positive. As discussed below, CLEO's success is partly due to the well-designed software communication interface between it and the underlying "Ygor" control system and partly due to the chosen technology.

2. CONTROL SOFTWARE ARCHITECTURE

Since the GBT monitor and control hardware and software system, "Ygor", is described in depth elsewhere (for example, see references 5-8 and http://www.gb.nrao.edu/GBT), I will concentrate here on the broad details of "Ygor" and how the software architecture of "Ygor" interacts with user interfaces.

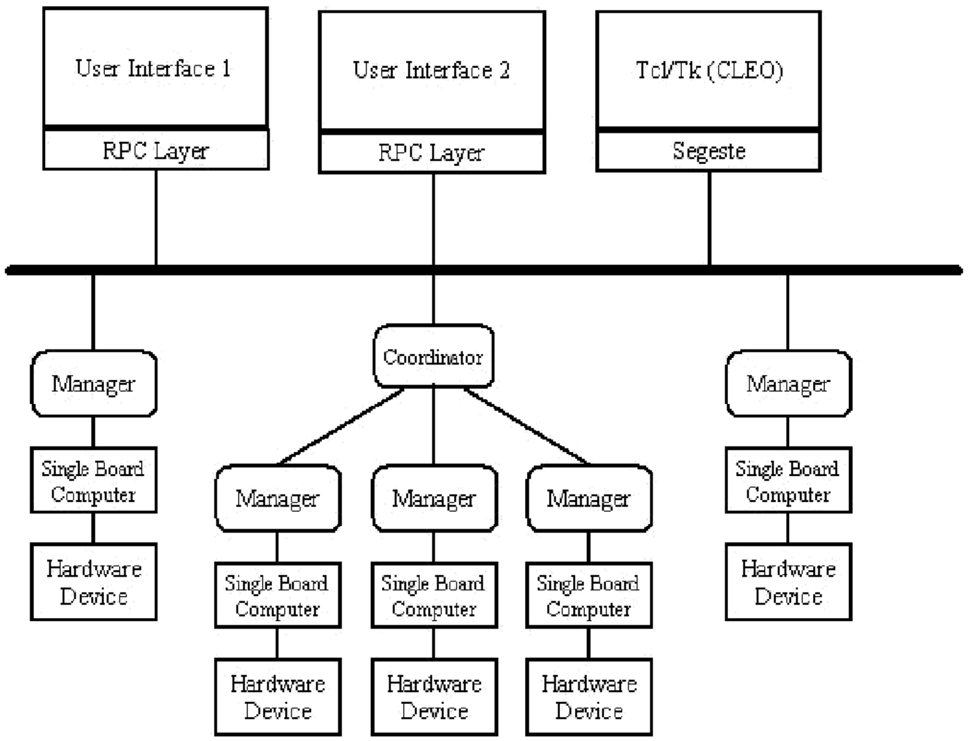


Figure 2: User interface and "Ygor" architecture. Note that three of the example managers are children of a coordinator while two other managers have no children. Two generic and the Tcl/Tk (CLEO) user interface technologies are diagrammed. Each technology requires its own RPC layer, which, in the case of CLEO, is called segeste.

The setup and operation of all devices, from I.F. electromechanical switches to the antenna servo system, are through software modules called managers. Managers are organized in a hierarchy where a parent manager coordinates children managers that have some common functionality. Most importantly for the builder of a user interface, every manager, and thus every device regardless of its design or use, presents to the builder of the user interface the same communication interface, the same commands and methods. Every manager has entities called parameters (for setting control values) and samplers (for monitoring) which, although the list of parameters and samplers may differ between managers, are always dealt with in the same fashion

In addition to managers, “Ygor”, through a separate communication protocol, provides messages that can be displayed by a user interface. Messages are strings of text that are generated throughout the “Ygor” programs to describe all types of expected and unexpected events. The “Ygor” system even provides to the user interfaces a database that describes the cabling between hardware components.

As shown in Figure 2, for every technology (Tcl, Perl, Java, etc.) in which a user interface is desired, a special but small software module needs to be created that translates the RPC calls used by the “Ygor” system into something which the technology understands. How this translation happens is completely transparent to the builders of the user interfaces. And, since the underlying communications are identical, usually a person adept at writing code in, for example, Perl and Tcl, can use his or her experience writing GBT interface code in Perl to write GBT interface code in Tcl.

The low cost for developing user interfaces for the GBT is partly a byproduct of this software architecture. In essence, the software architecture allows GUI builders to devote their time almost exclusively on how best to represent a particular control or monitor feature of the GBT without concerning themselves with the details of how to communicate with an individual device. Since all devices, from the antenna through receiver to detector, use the same software ‘methods,’ the heterogeneous GBT hardware looks to the GUI builder like a homogeneous class of objects. Only the ‘attributes’ of different devices are different.

Before I describe CLEO, I will outline the significant features of “Ygor” that user interfaces commonly exploit and which further help reduce development costs.

2.1 Manager commands

Every manager provides to a user interface the same set of commands. For example, a manager may be placed in *Off*, *Standby*, or *Ready* states by the commands **off**, **standby**, and **on** respectively. The commands **start** and **activate** cause the software to actually command the hardware. The command **start** causes an observation to begin while a**ctivate** causes the manager to do everything possible to get ready for an observation short of actually starting the observation. The commands **abort** and **stop,** where abort is a stronger version of stop, cause a manager to return to *Ready*.

Additionally, user interfaces can take advantage of certain introspective commands provided by every manager. For example, one can ask a manager to return its list of parameter names or sampler names. Managers provide to the user interfaces for every parameter and sampler a description that is usually about a sentence long, the range of legal values, and the units of a parameter (e.g., K, Volts, dBm). A typical use of this information by a user interface would be to create the text of the online help that accompanies a widget, to properly label a widget, or to provide error checking.

2.2 Manager Parameters

Each manager contains a set of parameters for controlling the underlying device. As far as a builder of a user interface is concerned, the only difference between managers is the set of parameters defined for that manager. To make matters even simpler, “Ygor” provides a subset of parameters that are built into all managers. For example, the *state* parameter describes the current state of a manager and may have the values of: *Off*, *Standby*, *Operating*, *Ready*, *Activating*, *Committed*, *Running*, *Stopping*, or *Aborting*. The status parameter summarizes the severity of the messages for the manager and any managers it coordinates and can be one of the following: C*lear, Info, Warning, Error, Fault,* or *Fatal*.

There are three types of parameters that a builder of a user interface might want to designate differently. They are:

- Control parameters are either set directly by the user or are computed from the values of other control parameters. They may be passed down to the hardware when an **activate** command is issued.

- Feedback parameters exist merely as a means for the manager to pass values back to the user. Their values may be set at any time by the system, but may not be modified by the user.
- Auto parameters, unlike Control parameters, are activated immediately upon any change of their value. They may only be set directly by the user and are for controlling aspects of devices that are observation independent. An example of an auto parameter is the rate at which a monitor point is sampled.

Parameters also share a common set of attributes that a user interface may want to display. Some of the important attributes of parameters are:

- *Touched* attribute indicates that a control parameter has been modified but that an **activate** command has not been issued and, thus, hasn't been acted upon or placed into the hardware.
- *Primary* attribute indicates a parameter that is part of the minimum set of parameters that fully define the state of the manager. Reverting all primary parameters to a previously saved set of values will always return the device to its previous configuration.
- *Illegal* attribute indicates a parameter whose value is illegal. The manager protects its associated device by checking all values and marking those having bad values as illegal.

2.3 Manager Samplers

Samplers are software "test points" that provide to the user interface a continuous stream of data values much like that from a power meter, frequency counter, or voltmeter. Sampler values are time-tagged and, like parameters, have a set of common attributes. For example, "Ygor" will set a sampler's error attribute whenever the sampler value is 'bad.'

2.4 Messages

Messages are essentially strings of text describing the nature of an event or problem within the system. Accompanying the message are attributes describing: (1) the severity of the event (e.g., Fatal, Fault, Error, Warning, Notice, or Information); (2) the name of the manager or device that generated the message; (3) a time stamp as to when the event occurred; (4) a time stamp when the event cleared; and (5) a message type (e.g., Asserted, Cleared, Transient, System Up, or System Down).

3. THE CONTROL LIBRARY FOR ENGINEERS AND OPERATORS

The "Control Library for Operators and Engineers," better known as CLEO, is the interface designed to meet the needs of staff operators, engineers and programmers in monitoring, controlling, and debugging hardware and software for the GBT. Although designed for operators and engineers, other NRAO staff members and observers have been finding CLEO very useful. CLEO provides a user interface that allows its users to modify almost every aspect of the GBT hardware. It provides tools that summarize the status of groups of systems as well as a wide suite of applications for easing the workload of those dealing with the GBT. Like all user interfaces, it was designed to bring clarity and ease of use to a complicated underlying system. CLEO is a system with extensive breadth and depth.

3.1 A success story: Tcl/Tk and CLEO

CLEO is written almost exclusively in Tcl/Tk[9], a scripting language similar to Perl and Python, and already used for control interfaces for various telescopes. For example, about a dozen papers describing telescope user interfaces that use Tcl/Tk were presented in "Telescope Control Systems III."[10] I was assisted in the project by the part-time efforts of Kevin Crump and Christine Rebinski, two undergraduates from Davis and Elkins College, a local liberal arts school. Both students had never programmed in Tcl/Tk before joining the project but, within a few weeks, were substantially contributing to the effort. In total, about eighteen months spread over four years have gone into the project. With almost 80,000 lines of code, our productivity averages to about 200 lines of code per day!

Although a significant part of this high productivity arose from the architecture of the underlying "Ygor" system, other factors came into play. One of the reasons Tcl/Tk was chosen was the potential for such a high productivity inherent in a scripting language like Tcl. As shown by Prechelt[11], scripting languages have about a factor of two cost advantage over traditional languages like C, C++, and Java simply because fewer lines of code are needed in a scripting language than in a traditional language. We concur since our estimate is that the project cost would have been somewhere between three and five times higher if we had used Java.

Another reason Tcl/Tk was chosen was the ease with which a scripting language could be learned, as was proven true by our experiences. Since Tcl/Tk has a large user base, we also exploited the large body of third-party libraries of debugging and development tools, most of which were free. In total, only about $150 was spent on programming tools.

With such a high rate of code production, one would start to worry that maintenance costs could be high if not enough effort was put into the system infrastructure. Yet, for over the past year, maintenance and development take up a mere 5% of my time. At most, one bug is reported every two weeks and bug turn-around time is usually less than a day. As I write this, only one bug remains outstanding. Thus, I believe that our high production rate did not compromise the maintainability of CLEO.

The success of a software project is only partly measured by costs and number of bugs. The user of the software must also be satisfied, which appears to be the case for CLEO. Currently, staff has submitted under a dozen, low-priority, low-cost enhancements that have yet to be implemented. Interviews indicate that engineers and operators like the system and observations show that seldom does a user falter when using CLEO. Other groups that use CLEO, such as astronomers, find more in CLEO that they wish were different. However, CLEO was not written for these occasional, accidental users and its design would have been compromised, and costs would have accelerated, if CLEO had tried to meet the needs of an extended audience.

3.2 CLEO architecture

CLEO isn't a single program but a set of over 40 applications. Dividing CLEO into multiple applications increased the reliability and robustness of the system without significantly increasing costs. Most applications are designed to work with a particular device, from the smallest of components to the telescope itself. Other applications provide summary information or gross control over a large set of devices. A few applications have nothing to do with the telescope or a device but instead help one use a system as powerful as CLEO.

Although we had a choice of writing CLEO using objected-oriented Tcl, we decided that development time would be less if we used a more procedural approach. Since most CLEO applications are well under two thousand lines of code, neither productivity nor maintainability significantly suffered by our decision. Only one application that deals with the display of messages would have benefited from an object-oriented approach.

The infrastructure behind CLEO is rather extensive with over half of our code residing in libraries. In addition to our libraries, CLEO makes heavy use of third-party software. The following lists those libraries I think would be helpful to other developers of Tcl programs:

- BLT: a widget and command library developed by G. Howlett and especially useful for generating histograms and graphs (http://incrtcl.sourceforge.net/blt).
- vTcl: a GUI builder and integrated development environment for Tcl/Tk developed by S. Allen and written entirely in Tcl/Tk (http://www.giantlaser.com/vtcl).
- Bwidget: a mega-widget library developed by UNIFIX (http://sourceforge.net/projects/tcllib)
- tkCon: a combined debugger and console developed by J. Hobbs (http://tcl.activestate.com/community/hobbs).
- Freewrap: a utility developed by D. LaBelle that creates standalone Tcl applications for distribution to users (http://freewrap.sourceforge.net).

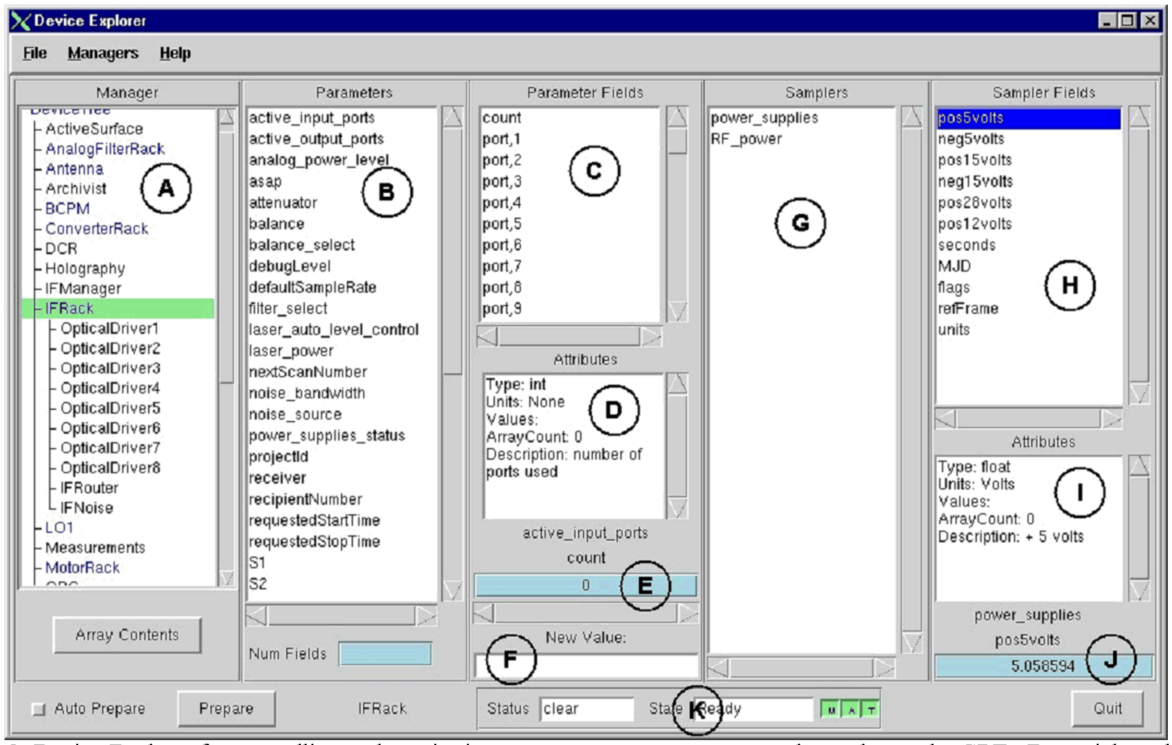
Since Tcl is a scripting language, debugging is both easier and more difficult than with traditional languages like C or Java. Since coding errors are only found during run time, a developer must test every possible consequence of conditional and loop constructs. To achieve this level of testing, every CLEO application uses a generic 'simulator' that allows the developer to test a significant fraction of the application-specific code. Some of the more complicated applications require special simulators.

Debugging Tcl is simplified because of tools like tkCon mentioned above. With tkCon, if a bug is found, one can actually modify and test the code while the application is still running. There is no need to exit that instance of the application, start up a debugger, restart the program, and try to recreate the condition that generated the problem. Additionally, if a run-time error occurs (e.g., a reference to a variable that doesn't exist), CLEO usually catches the error and automatically e-mails the developer a full stack trace of the problem. In essence, CLEO provides a description of a bug it has discovered within itself! These features substantially reduce the time needed for bug fixes, create more robust and reliable code, and more than compensate for the need for simulators.

3.3 CLEO applications

Space prevents me from discussing more than a few of the CLEO applications. The following is a collection of some typical applications as well as those that designers of control software might find interesting.

3.3.1 Device Explorer



Type: int
Units: None
Values:
ArrayCount: 0
Description: number of ports used

active_input_ports
count
0

Type: float
Units: Volts
Values:
ArrayCount: 0
Description: + 5 volts

power_supplies
pos5volts
5.058594

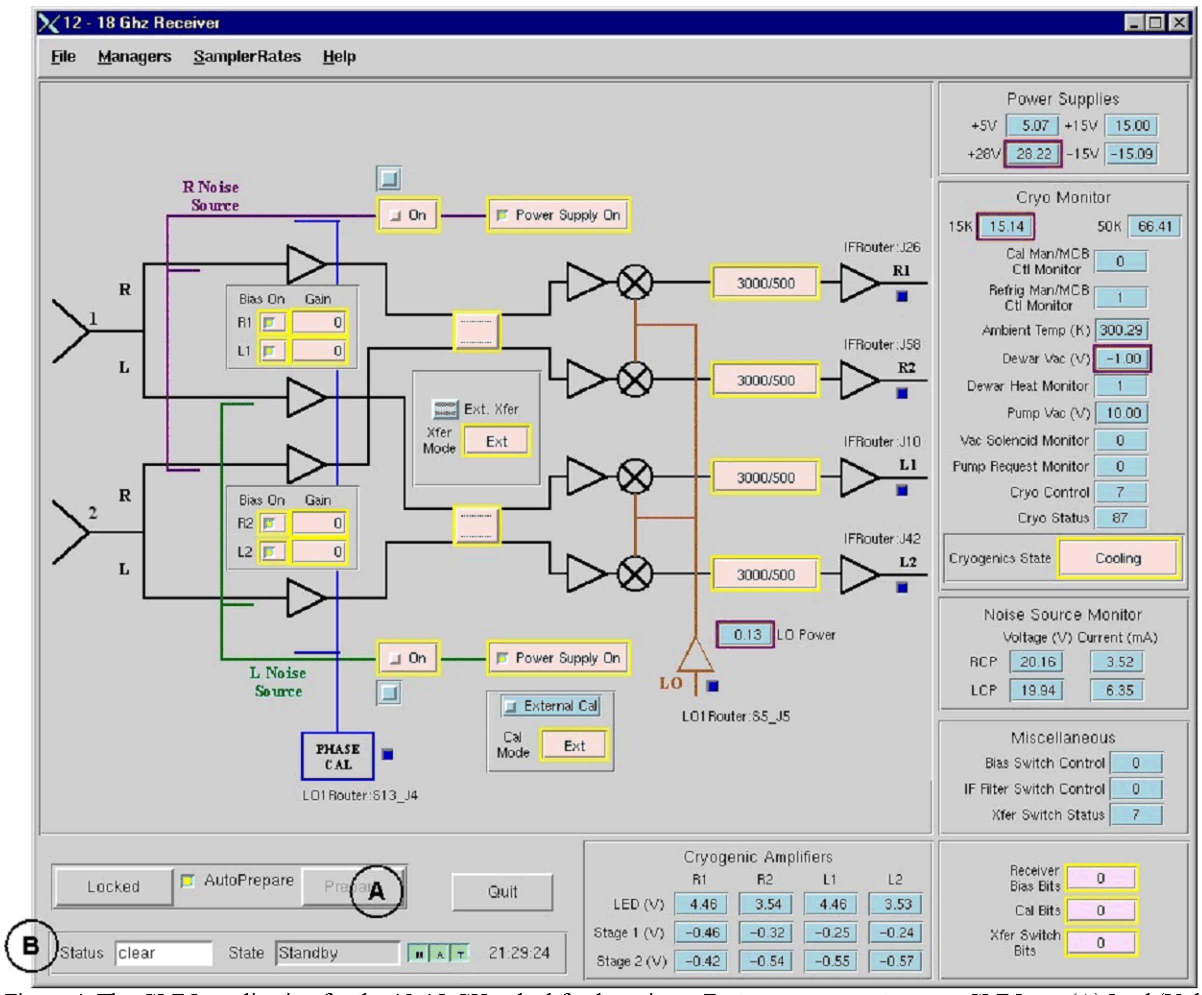
Status clear
State Ready

Figure 3: Device Explorer for controlling and monitoring every manager, parameter, and sampler on the GBT. From right to left the important parts of the screen are: (A) a hierarchal list of coordinators and managers, (B) the list of parameters for the selected manager, (C) the fields within a parameter, (D) the attributes of the selected parameter, (E) the current value of the parameter, (F) a widget for entering a new value for the parameter, (G) list of samplers for the selected manager, (H) the fields within a sampler, (I) the attributes of the sampler, (J) the current value of the sampler, and (K) the state and status of the selected manager.

Device Explorer was the first CLEO application we presented to our users, a decision that was well-calculated since simpler applications could have been released first. Device Explorer (Figure 3) allows the user to interact with every coordinator, manager, parameter, and sampler in the system. In its design we exploited the commonality of methods provided by "Ygor" for every manager. With the introduction of Device Explorer, and for the first time, engineers and programmers could modify every aspect of a device without requiring that someone create a user interface for that device. In essence, by releasing Device Explorer we immediately relieved the pressure for manager-specific GUIs. Since its release, Device Explorer remains one of the most used and most clever CLEO applications.

3.3.2 Typical device screens

Engineers requested that screens for devices like receivers, L.O., and I.F. equipment use schematics to show the flow of signals through the device. Figure 4 is the screen for the 12-18 GHz, dual-feed receiver and is typical of many CLEO applications. Note the placement of widgets on top of a schematic that we created as a GIF file using SmartDraw (http://www.smartdraw.com), a substantially easier task than if we had used the alternative of drawing the schematic using native Tk canvas commands.



Power Supplies

+5V	5.07	+15V	15.00
+28V	28.22	-15V	-15.09

Bias On / Gain: R1 0, L1 0, R2 0, L2 0

LO Power 0.13

Cryo Monitor

15K	15.14
50K	66.41
Cal Man/MCB Ctl Monitor	0
Refrig Man/MCB Ctl Monitor	1
Ambient Temp (K)	300.29
Dewar Vac (V)	-1.00
Dewar Heat Monitor	1
Pump Vac (V)	10.00
Vac Solenoid Monitor	0
Pump Request Monitor	0
Cryo Control	7
Cryo Status	87
Cryogenics State	Cooling

Noise Source Monitor

	Voltage (V)	Current (mA)
RCP	20.16	3.52
LCP	19.94	6.35

Miscellaneous

Bias Switch Control	0
IF Filter Switch Control	0
Xfer Switch Status	7

Cryogenic Amplifiers

	R1	R2	L1	L2
LED (V)	4.46	3.54	4.46	3.53
Stage 1 (V)	-0.46	-0.32	-0.25	-0.24
Stage 2 (V)	-0.42	-0.54	-0.55	-0.57

Receiver Bias Bits	0
Cal Bits	0
Xfer Switch Bits	0

Status clear State Standby 21:29:24

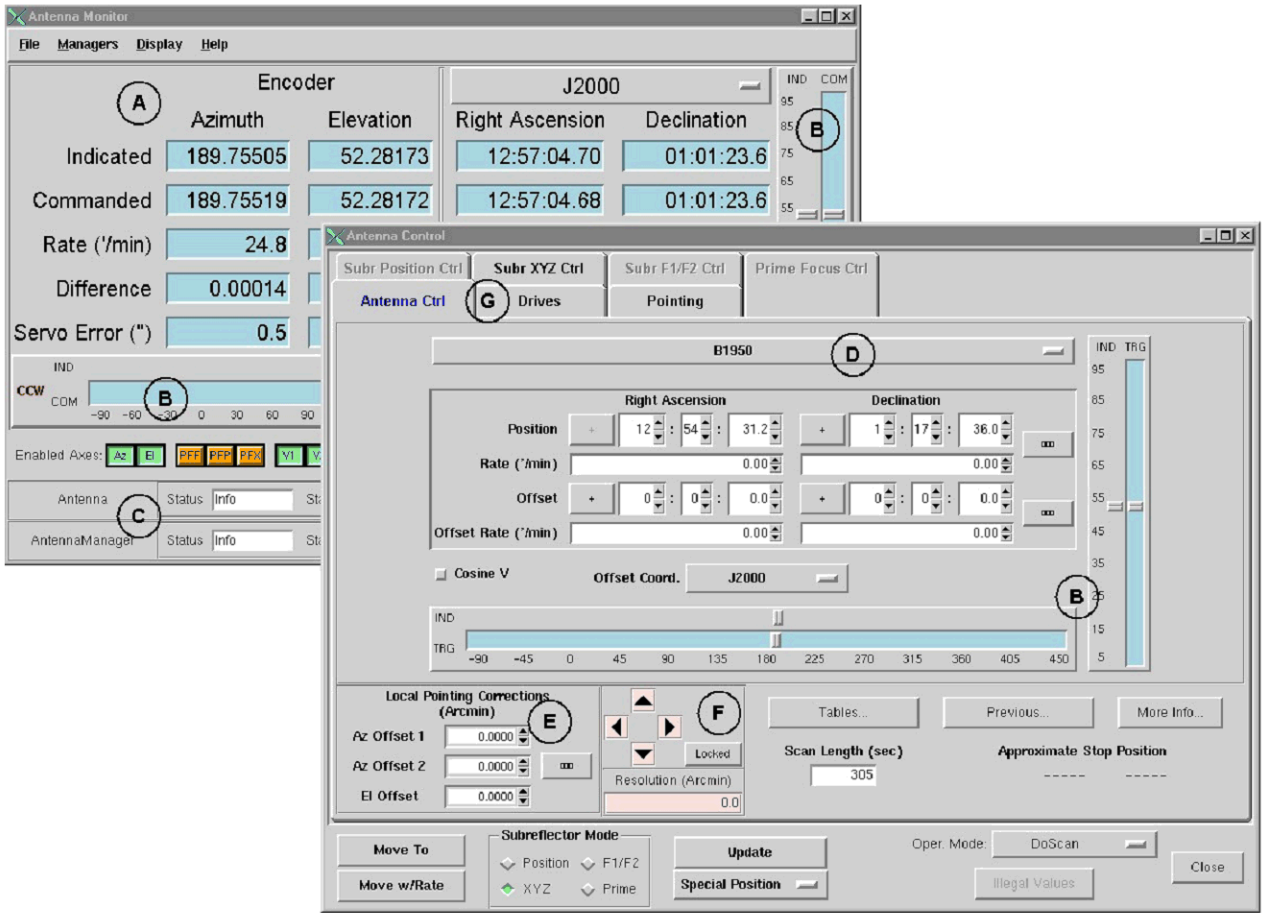
Figure 4: The CLEO application for the 12-18 GHz, dual-feed receiver. Features common to many CLEO are (A) Lock/Unlock frame of widgets and (B) a State/Status frame of widgets.

Device screens typically have a "Lock/Unlock" frame of widgets. Since any CLEO application can be started by anyone, engineers didn't want unauthorized users controlling devices. A typical CLEO application starts in a locked state which has all control widgets deactivated. A user must be in a security database, described below, to unlock a screen and control a device. Thus, everyone can monitor every device but only a select few can control a device.

Screens usually have a "State/Status" frame of widgets (a reuse of the similar frame in Device Explorer) that depicts and color-codes the state and status of the managers associated with the screen. Widget background colors differentiate widgets associated with samplers, control parameters, and feedback parameters. The color of widget borders shows, for example, whether a sampler has a possible problem, whether a parameter value has been sent to the hardware, or whether a parameter value is illegal.

3.3.3 CLEO antenna screen

Figure 5 shows the CLEO antenna screen, one of the more complicated applications but still very similar in design to most other CLEO applications. Here, the job of controlling and monitoring the telescope is divided between different screens. Tab notebooks are heavily used as a way of organizing functionalities as well as preventing an overloading of widgets on a single screen. Note that the color-coding here is the same as used everywhere else in CLEO, which gives every CLEO application the same look and feel.



	Encoder Azimuth	Encoder Elevation	J2000 Right Ascension	J2000 Declination
Indicated	189.75505	52.28173	12:57:04.70	01:01:23.6
Commanded	189.75519	52.28172	12:57:04.68	01:01:23.6
Rate ('/min)	24.8			
Difference	0.00014			
Servo Error (")	0.5			

Antenna Status Info
AntennaManager Status Info

B1950

	Right Ascension	Declination
Position	12 : 54 : 31.2	+ 1 : 17 : 36.0
Rate ('/min)	0.00	0.00
Offset	+ 0 : 0 : 0.0	+ 0 : 0 : 0.0
Offset Rate ('/min)	0.00	0.00

Cosine V
Offset Coord. J2000

Local Pointing Corrections (Arcmin)
Az Offset 1 0.0000
Az Offset 2 0.0000
El Offset 0.0000

Resolution (Arcmin) 0.0
Scan Length (sec) 305
Approximate Stop Position ----- -----

Subreflector Mode: Position, F1/F2, XYZ [✓], Prime
Special Position
Oper. Mode: DoScan

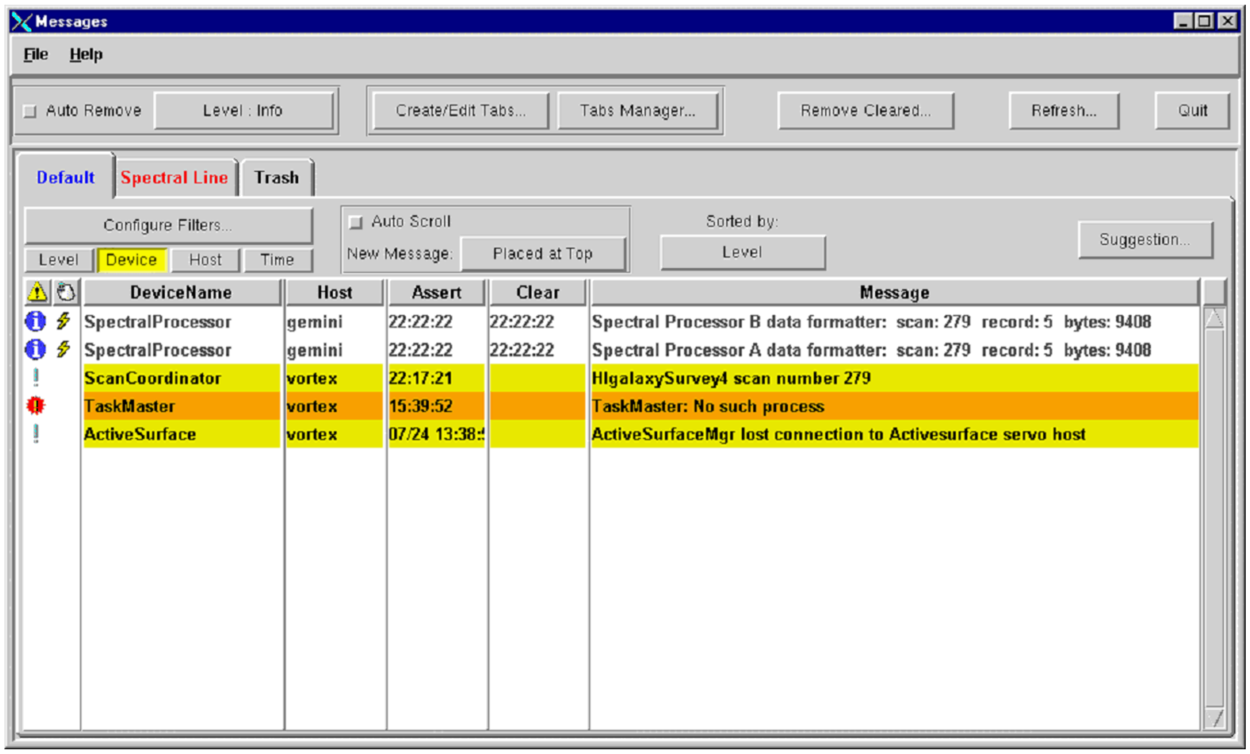
Figure 5: CLEO Antenna monitor screen (upper left), control screen (right).. Important aspects of this the screens are: (A) current telescope and commanded positions depicted numerically and (B) graphically relative to the limit switches, (C) State/Status frame, (D) widgets for commanding the telescope to a position in a desired coordinate system, (E) pointing corrections (F) software "joy stick" (G) tab notebook for accessing the controls for the telescope optics, drives, etc.

3.3.4 CLEO message application

By far the most complicated CLEO application is that used to display messages from devices and managers (Figure 6). The application provides the ability to filter messages by time, device, and message severity. Messages can be sorted by severity, time, text, and assert or clear times. Messages can be organized by topics by dragging and dropping them onto user-defined tab notebooks. A color scheme, similar to that used by the rest of CLEO, depicts message severity. There is even a tool for submitting suggested changes to the text of error messages.

The message application allows for the generation of sounds using the CLEO sound server (see §3.3.8) when certain events occur. It also provides for message 'inference' and 'inhibition,' both of which are aimed at reducing a cascade of error messages from a single failure. For example, if a power supply fails, one can expect other aspects of the associated device to fail, possibly producing a cascade of error messages. If the power supply is being monitored, then message 'inhibition' will display the message for the power supply and inhibit the display of the messages from all the other components whose failure resulted from the bad power supply. However, if the power supply isn't being monitored, one can infer from the resulting cascade of error messages that the power supply was at fault; the message application hides all messages and generates its own 'inferred' message concerning the power supply.

CLEO provides a separate utility that searches the log of messages for messages that satisfy a user-specified set of criteria. The application then graphically displays the found messages versus the time the messages occurred. This "Analyze Messages" application has proved to be very useful in debugging problems after the fact and to catch patterns in the sequence of events that produce messages.

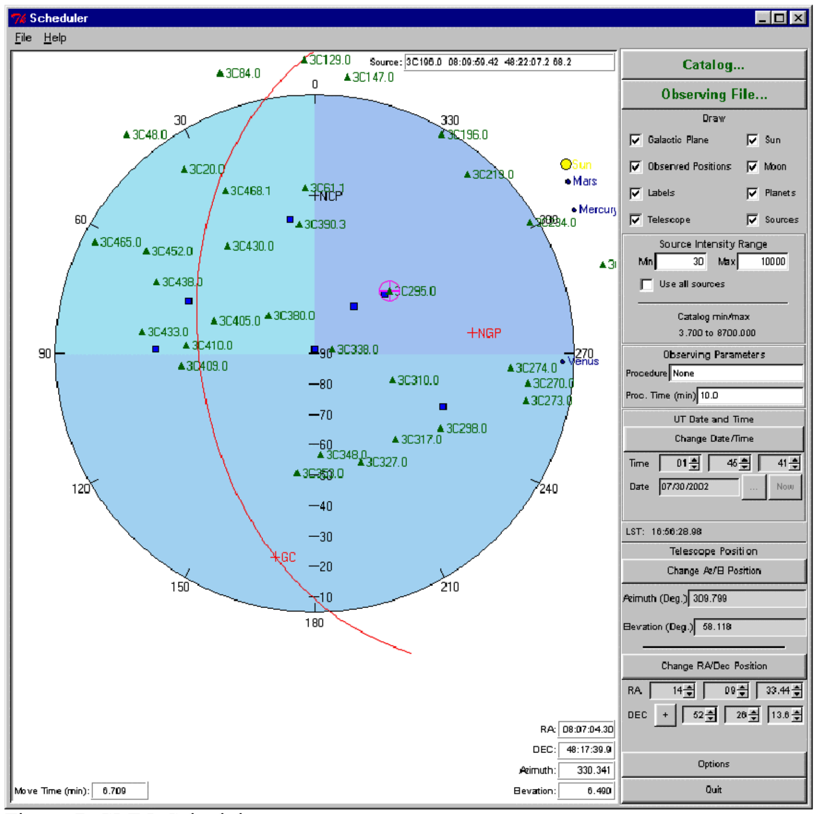


⚠		DeviceName	Host	Assert	Clear	Message
ℹ	⚡	SpectralProcessor	gemini	22:22:22	22:22:22	Spectral Processor B data formatter: scan: 279 record: 5 bytes: 9408
ℹ	⚡	SpectralProcessor	gemini	22:22:22	22:22:22	Spectral Processor A data formatter: scan: 279 record: 5 bytes: 9408
!		ScanCoordinator	vortex	22:17:21		HIgalaxySurvey4 scan number 279
●		TaskMaster	vortex	15:39:52		TaskMaster: No such process
!		ActiveSurface	vortex	07/24 13:38:		ActiveSurfaceMgr lost connection to Activesurface servo host

Figure 6: The CLEO message application

3.3.5 CLEO scheduler

A common task for staff and observers is the scheduling of a series of observations. The CLEO "scheduler" provides a graphical interface for converting a catalog of source positions into an observing file. It is a standalone application that can be used with almost any altitude-azimuth mounted telescope. As shown in Figure 7, the application shows a representation of the celestial sphere, the location on the sky of the objects in the catalog, and a marker depicting the position of a virtual telescope. The user specifies the duration of observations and then starts clicking on sources to generate an observing file. When a user clicks on a source, the application calculates the time it will take to move the virtual telescope to the source (properly corrected for the current telescope cable wrap) and perform the observation. The source and telescope positions are then updated to the time at the end of the observation. The user then clicks on source after source until the observing file is completed.



Source: 3C196.0 08:09:59.42 48:22:07.2 68.2

Min 30 Max 10000

Catalog min/max
3.700 to 8700.000

Procedure None
Proc. Time (min) 10.0

Time 01 46 41
Date 07/30/2002

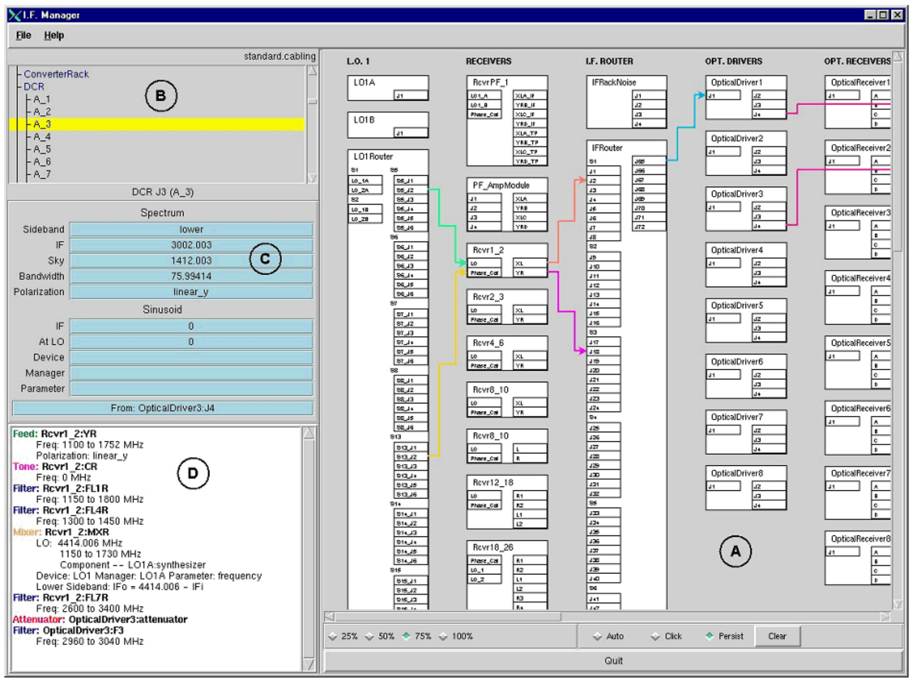
LST: 16:56:28.98

Azimuth (Deg.) 309.799
Elevation (Deg.) 58.118

RA 14 09 33.44
DEC + 52 26 13.6

RA: 08:07:04.30
DEC: 48:17:39.9
Azimuth: 330.341
Elevation: 6.490
Move Time (min): 6.709

Figure 7: CLEO Scheduler



DCR J3 (A_3)

Spectrum

Sideband	lower
IF	3002.003
Sky	1412.003
Bandwidth	75.99414
Polarization	linear_y

Sinusoid

IF	0
At LO	0
Device	
Manager	
Parameter	

From: OpticalDriver3:J4

```
Feed: Rcvr1_2:YR
        Freq: 1100 to 1752 MHz
        Polarization: linear_y
Tone: Rcvr1_2:CR
        Freq: 0 MHz
Filter: Rcvr1_2:FL1R
        Freq: 1150 to 1800 MHz
Filter: Rcvr1_2:FL4R
        Freq: 1300 to 1450 MHz
Mixer: Rcvr1_2:MXR
        LO:  4414.006 MHz
             1150 to 1730 MHz
             Component -- LO1A:synthesizer
        Device: LO1 Manager: LO1A Parameter: frequency
        Lower Sideband: IFo = 4414.006 - IFi
Filter: Rcvr1_2:FL7R
        Freq: 2600 to 3400 MHz
Attenuator: OpticalDriver3:attenuator
Filter: OpticalDriver3:F3
        Freq: 2960 to 3040 MHz
```

Figure 8: IF Manager application. The important parts of the screen are: (A) a graphical representation of the cabling between selected devices, (B) a hierarchal list of all connection points in the system, (C) a summary of the signal processing upstream from some selected point in the system, and (D) a full description of the signal processing.

3.3.6 IF Manager

The GBT probably possesses one of the most complicated systems of receivers, local oscillators, and signal processing hardware ever used on a radio telescope. To help staff and observers properly set up the system we have developed the CLEO IF Manager, shown in Figure 8. Its job is to represent the cabling between devices as well as all of the signal processing that has happened from the receiver to any point in the system. Using this tool, one can discern whether or not the system has been properly configured as well as help determine where a problem might exist.

3.3.7 CLEO Server

Since the GBT resides in a rural area where some support staff live up to an hour from the observatory, engineers requested that they should be able to run CLEO applications from their home computers so as to reduce the response time to simple problems. Unfortunately, the distances between homes and telephone switching stations, as well as the low population densities, means that engineers typically would be using modems with at best 26k-baud connections. Thus, sending, for example, X-windows displays to a home PC running an X-windows server is impractical. The CLEO Server, currently undergoing beta testing, should make this request a reality.

Every CLEO application can run just as easily on a Windows-based PC as well as on various flavors of Unix. If an engineer were to run a CLEO application from hom,e all that is required is some way of providing that CLEO application with a path into the "Ygor" managers. The CLEO Server provides that path with each remote computer acting as a client to the server. To get the most out of the limited connection speed, the server formats and truncates all parameter and sampler values using formatting specifications provided by each client. Also, the server keeps track of the last formatted values sent to a client and, if the formatted value hasn't changed, does not resend the value.

Our testing indicates that the system works and that a 26k-baud connection can handle up to ten typical CLEO applications simultaneously. We expect to distribute CLEO to home users within the next two months.

3.3.8 Other CLEO applications

Some of the other interesting applications that make up CLEO include:

Launcher: The Launcher is usually the first CLEO application a user starts. Because the suite of CLEO applications is rather large, the Launcher's job is to provide a simple and organized interface for starting other CLEO applications. Since the telescope operator sits in front of multiple computers, each of which might have multiple screens, the Launcher also provides the capability of starting any CLEO application on any of these monitors.

Gateway: One of the requirements for CLEO was that a staff member should be able to monitor a device but only the telescope operator and one other person, (e.g., an observer or engineer) can control the device. To provide this level of security, CLEO provides a password-protected application, called the Gateway, which maintains a database describing who has the ability to control what device. Both "Ygor" and CLEO use this database to determine control privileges.

Sound Server: The Sound Server addresses the problem of what to do if multiple CLEO applications running on multiple computers on a myriad of screens want to generate sounds associated with events. Every CLEO application can act as a client of the Sound Server and, when a sound is needed, provides the Sound Server the name of the sound file to play, a priority level for the sound, and the number of times the sound is to be repeated. The client can also specify that the sound is to be played continuously until the user acknowledges the event. The server then queues the request for sounds based on arrival time and priority and plays the sound at the front of the queue.

Save/Load Configuration: The Save Configuration application allows the user to store for later use the values of the primary parameters for a user-selected set of devices. The Load Configuration application uses a previously saved configuration file to return the system back to the state it was in when the configuration was saved.

Manager Control: A single application that displays the state and status of every manager in the system. With one glance, a user can determine the health of every device as well as what that device is doing.

4. CONCLUSIONS

The success of CLEO can be measured in terms of low development and maintenance costs, reliability, robustness, and user satisfaction.

- Low development costs are promoted by the uniformity of methods provided by the underlying control and monitor system. Every device, regardless of what it does, looks the same and employs the same methods for specifying commands, altering parameters, and accessing sampled monitor data. Only the attributes of managers differ.
- The technology of scripting languages, in particular Tcl/Tk, and the ability to tap into a wide range of existing software libraries, reduces development costs as well as maintenance costs.
- The robustness and reliability of CLEO is due to a combination of using simulators to test scripts and sophisticated debugging tools that promote finding and fixing bugs in a timely and relatively painless fashion.
- Robustness and reliability were improved by dividing CLEO into a suite of applications instead of developing a single application.
- User satisfaction stems from a number of reasons. The design adheres to the requirements set by its users and follows common practices in GUI design. Users will also appreciate a system that was released before they needed it, is robust and reliable, and whose infrastructure allows for fixing bugs quickly and allows for adding new features.

5. ACKNOWLEDGMENTS

The National Radio Astronomy Observatory is a facility of the National Science Foundation, operated under cooperative agreement by Associated Universities, Inc.

REFERENCES

1. vanden Bout, P., “Green Bank Telescope and the Millimeter Array,” Proc. SPIE Vol. 3357, p. 2-11, Advanced Technology MMW, Radio and Terahertz Telescopes, Thomas G. Phillips; ed., 1998.
2. Lockman, F. J., “The Green Bank Telescope: an Overview,” Proc. SPIE Vol. 3357, p. 656-665, Advanced Technology MMW, Radio and Terahertz Telescopes, Thomas G. Phillips; ed., 1998.
3. Jewell, P. R., “The Green Bank Telescope,” Proc SPIE Vol. 4015, p. 136-147, Radio Telescopes, Harvey R. Butcher; ed., 2000.
4. Lacasse, R., “Green Bank Telescope Active Surface System,” Proc. SPIE Vol. 3351, p. 298-310, Telescope Control Systems III, Hilton Lewis, ed., 1998.
5. Brandt, J.J., “Controlling the Green Bank Telescope,“ Proc. SPIE Vol. 3351, p. 96-108, Advanced Telescope and Instrumentation Control Software”, Hilton Lewis, ed., 2000.
6. Clark, M. H., “The Control Software Architecture for the Green Bank Telescope,” Proc. SPIE Vol. 3351, p. 287-296, Telescope Control Systems III, Hilton Lewis, ed., 1998.
7. Ford, J.M., “GBT Telescope and instrumentation control system hardware architecture: computers, networks, interfaces, and timing,” Proc. SPIE Vol. 3351, p. 387-395, Telescope Control Systems III, Hilton Lewis, ed., 1998.
8. Brandt, J.J., “Reliable Multicast Protocols and their Application on the Green Bank Telescope,” Proc. SPIE Vol. 3351, p. 396-405, Telescope Control Systems III, Hilton Lewis, ed., 1998.
9. Ousterhout, J.K., “Tcl and the Tk Toolkit,” Addison-Wesley, 1994
10. “Telescope Control Systems III,” Proc. SPIE Vol. 3351, p. 190-196, Hilton Lewis, ed., 1998.
11. Prechelt, L., “An empirical comparison of C, C++, Java, Perl, Pyhton, Rexx and Tcl,” IEEE Computer 33(10), 23-29, 2000.