SOAR Telescope Control System: A Rapid Prototype and Development in LabVIEW

Michael C. Ashe [a] , Germán Schumacher [b]

[a]Imaginatics, 943 North Rd, Groton, CT 06340

[b]National Optical Astronomy Observatories, P.O. Box 26732, Tucson, AZ 85726-6732

ABSTRACT

Most large telescope projects are using UNIX, C or C++ and VME hardware for development of their TCS. The SOAR Telescope is using the LabVIEW graphical programming environment under Real-time Linux and WindowsNT, hosted on PCI-Intel based hardware to achieve a flexible, cost effective and reusable TCS architecture.

A Rapid Prototype and full development plan of the SOAR TCS is reviewed to show advances in: (1) Prototyping speed, which makes implementation and test of features faster than specification under older methods. This allows the development environment and prototype modules to become partners with and part of the specification documents. (2) Real-Time performance and reliability through use of RT Linux. (3) Visually Rich GUI development that allows an emphasis on "seeing" versus "reading". (4) Long-Term DataLogging and Internet subscription service of all desired variables with instant recall of historical trend data. (5) A "plug-in" software architecture which enables rapid reconfiguration and reuse of the system and/or plug-ins utilizing LabVIEW graphical modules, a scripting language engine(in LabVIEW) and encapsulation of interfaces in "instrument-driver" style "plug-in" modules. (6) A platform-independent development environment and distributed architecture allowing secure internet observation and control via every major OS and hardware platform.

Keywords: SOAR, LabVIEW, Linux, Real-Time, Telescope Control System, development environment.

1. INTRODUCTION

The SOuthern Astrophysical Research (SOAR) telescope is a project to build a 4.2-meter telescope on Cerro Pachon in Chile. The project is a collaboration between UNC, Brazil, Michigan State University, and the National Optical Astronomy Observatories (NOAO), (with Chile as host nation) with the objective to design, construct, and commission a 4-meter optical/IR telescope that is optimized for quality. It will be the largest 4-meter class telescope in the world.

The basic design is an F/16 Ritchey-Chretien with extremely challenging specifications. The telescope will feature an active optical system that includes a figure-controlled primary mirror (M1), an actively aligned secondary mirror (M2), fast tip/tilt image stabilization at the tertiary mirror (M3), an optimization wave-front sensor, and control electronics and software." [1]

Instruments will be located on two Instrument Selector Boxes (ISB) at the Nasmyth points and at three bent-cassegrain points. The dome features a single shutter with associated windscreen, several fan vents and a chilled water HVAC system for environmental control. A weather tower will report environmental conditions outside the dome. Internal temperature, humidity and other variables will be monitored and controlled to optimize isothermal conditions.

The hardware is partitioned into five main subsystems: the Mount Subsystem, the Adaptive Optics Subsystem, the Dome Subsystem, the Instrument Adapter Subsystem, the Environmental Subsystem.

The software is partitioned into four main systems: the Observatory Control System (OCS), the Telescope Control System (TCS), the DataLogger System and the Instrument Control System (ICS). **This paper focuses on the TCS and DataLogger**.

The SOAR project office desired to take advantage of recent technical advances and cost reductions in commercial PC based hardware and software. An evaluation of several tools lead to choosing LabVIEW as the primary candidate. LabVIEW is a proven programming system and is being used in science, engineering and factory process control all over the world. The SOAR project team decided to implement a design study, with a prototype LabVIEW TCS as a deliverable, prior to committing to use LabVIEW in the full TCS development. A search for LabVIEW integrators lead to the selection of Imaginatics to implement the prototype. In

addition, the ability to connect with slalib 'C' code, in a non-trivial form, (such as the Gemini kernel) was deemed critical to the approval of LabVIEW in the final TCS. [2]

The SOAR Instrumentation Control (IC) committee has also selected the LabVIEW as the standard environment for all SOAR instrument related components. Commonality of language and some of the underlying libraries will make long term maintenance easier and more cost effective for both TCS and Instrument Control System (ICS).

This lead to a series of meetings and prototype revisions of increasing complexity and capability. Currently, two levels of prototype are implemented. The third level is nearing completion, and will include most of the final graphical user interfaces (GUI). The first "demo" prototype was developed at a meeting of SOAR, Imaginatics and Rutherford Appleton Labs (RAL) in September of 1998.

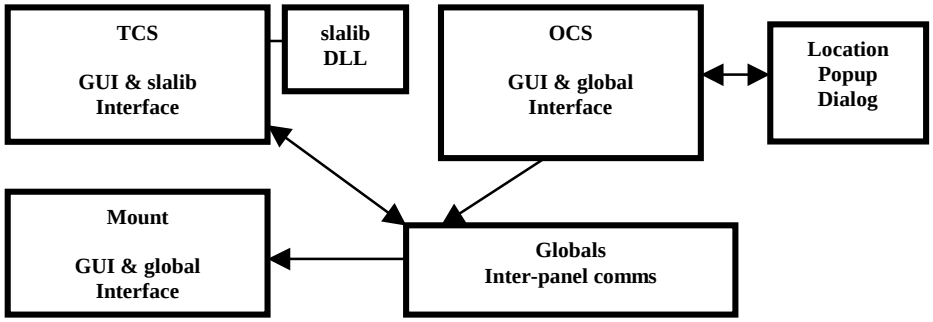

Figure 1.0 First TCS demo architecture for Slalib test. The module layout of the first TCS demo implemented in LabVIEW and "C" for the SLALIB function calls.

The purpose of the demo was to introduce RAL personnel to LabVIEW, to show LabVIEW's Rapid Application Development (RAD) capabilities, to demonstrate the capability to interface to pre-written C libraries, (specifically SLALIB) and to enable a discussion on what features were required to be demonstrated in the SOAR TCS Prototype I (STP-1) version. The demo included modules for OCS entry of target RA/DEC, TCS calculation of associated Alt/Az position demands with timestamps using calls to SLALIB routines, Mount reception of demanded position with graphing of time between updates. Initial communications were handled using global variables because the demo was hosted on a single CPU running Windows98.

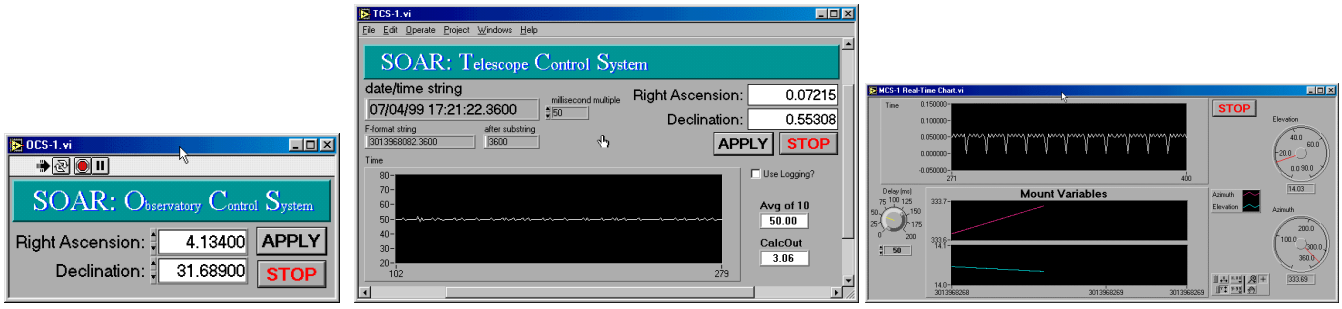

Figure 2.0 First demo GUIs. The module layout of the first TCS demo implemented in LabVIEW and "C" for the SLALIB function calls.

The demo showed the required capabilities with the exception of stability of timing in the 20 Hz update rate. This can be observed in the center and right side graphs in figure 2. The irregularity of timing on the Windows platform was a concern to all. Implementation of several new capabilities in the STP-1 version of the TCS were requested to fully demonstrate LabVIEW's capability to handle the robustness, timing and flexibility requirements of SOAR: (1) Distributed subsystems. (2) Show slow, medium and fast kernel loops. (3) Incorporate a modified Gemini kernel. (4) Show subsystem status, health and if in position. (5) Show quick initialization and rebooting. (6) Incorporate logging of subsystem data. (7) Simulate Mount servo action/response. (8) Develop an OCS GUI interface. (9) Incorporate configuration files for initialization. (10) Interpolation of the 20 Hz commands. (11) Improve smoothness and regularity of 20 Hz position demands. (12) Implement the Kernel/TCS on a Linux platform. (13) SkyMap Interface to OCS to demonstrate utilization of commercial software that accesses the professional star catalogs.

Imaginatics was tasked to develop the non-kernel portion of the STP-1. RAL was tasked to develop a modified Gemini kernel in "C“ code, along with instructions in its operation. This kernel was to demonstrate LabVIEW and PC capability to handle the number crunching load of a real kernel.

The STP-1 subsystems included: Mount, Dome, Adaptive Optics, Instrument Adapter and Environmental simulations, all on PCs/notebooks connected by TCP/IP and 10BaseT Ethernet. The simulations had GUIs that represented the subsystems in a pictorial “mimic bus” manner, demonstrating LabVIEW’s capabilities in user interface development.

The TCS Prototype I (STP-1) implementation: The software developed during this phase of the project was divided into seven main modules, per the layout in Figure 3. It performed all of the required functionality including the star catalog interface (using SkyMap). The modules are:

OCS GUI & SkyMap Interface: Astrometric target data entry and routing to the TCS module.

TCS GUI & DataLogger GUI: Astrometric data (from OCS or front panel). Calling of the Gemini kernel at regular intervals. Supplying Alt-Az commands to the Mount and Dome from the Kernel. Commands to other subsystems and logging of return data.

Adaptive Optics Subsystem Simulator GUI: Display of M1 figure positioners and M2 position with 5 degrees of freedom.

Mount Subsystem Simulator GUI: Acceptance of Alt-Az commands and simulation of motor slewing and tracking. Rotator position.

Environmental Subsystem Simulator GUI: Display and simulated control of two chillers with associated pumps and three fans each. Also includes weather station information and an expandable table of temperature and other industrial signals, such as valve position, pressure, strain, etc.

Dome Subsystem Simulator GUI: Simulation of Dome azimuth, two shutter doors, two vents, and a windscreen.

Instrument Adapter Subsystem Simulator GUI: Selection of Instrument Selector Box position and instrument type.

The module layout for STP-1 was implemented as follows:

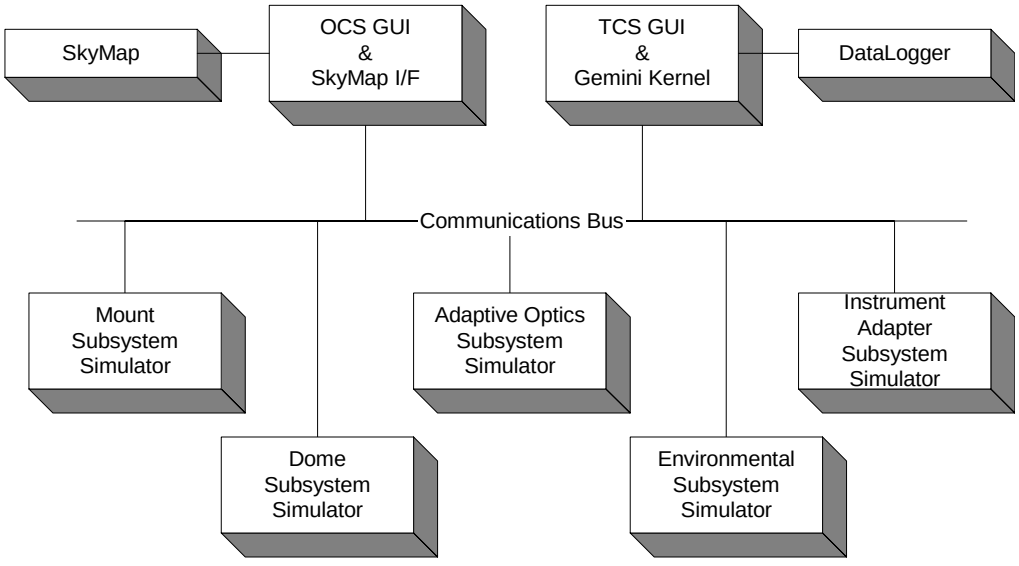

Figure 3.0 TCS Prototype I software module layout. This implementation is using 10baseT Ethernet and LabVIEW’s built in TCP/IP for all communications. This interconnectivity will be replicated in later prototypes and the final TCS implementation, although the physical layer shall be 100baseT or faster.

The subsystem simulators all included controls for setting local control or remote control modes. Each had a “Health” indicator, which was manually set from the front panel and was then transmitted to the TCS for display and logging. The Mount and Environmental simulators were developed first, as they were the most complex. These were then used as templates to make the other subsystem simulators. Each simulator acted as a TCP/IP server to the TCS module, which acted as the client.

Software reuse:

Rapid development of the different simulators gave a good demonstration of the efficiency of LabVIEW development. As soon as the code was finished for the first modules it was copied under a different name for reuse on this project

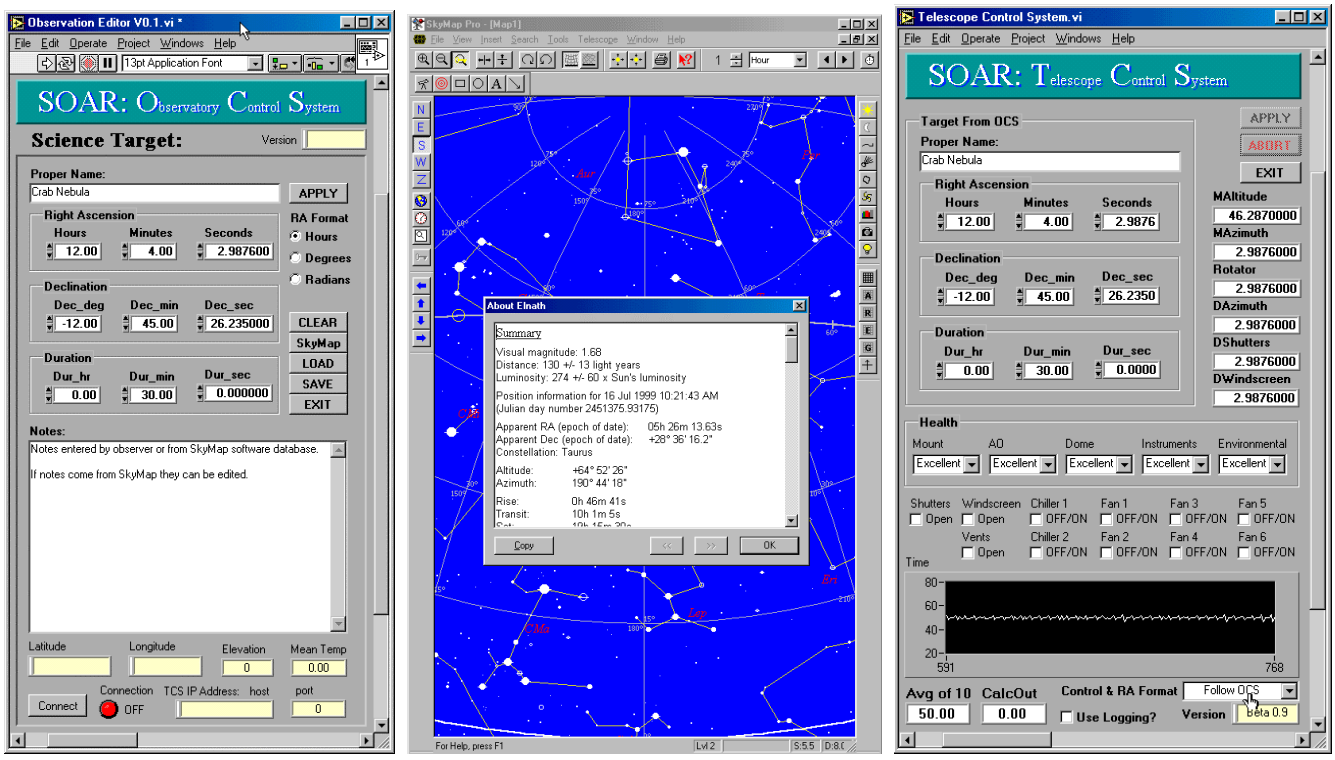

Figure 4.0 OCS, SkyMap and TCS GUIs for STP-1. Many of the custom controls developed in the TCS GUI were reused in the OCS and other GUIs/modules. The SkyMap interface was implemented using the windows clipboard and clipboard manipulation VIs in LabVIEW.

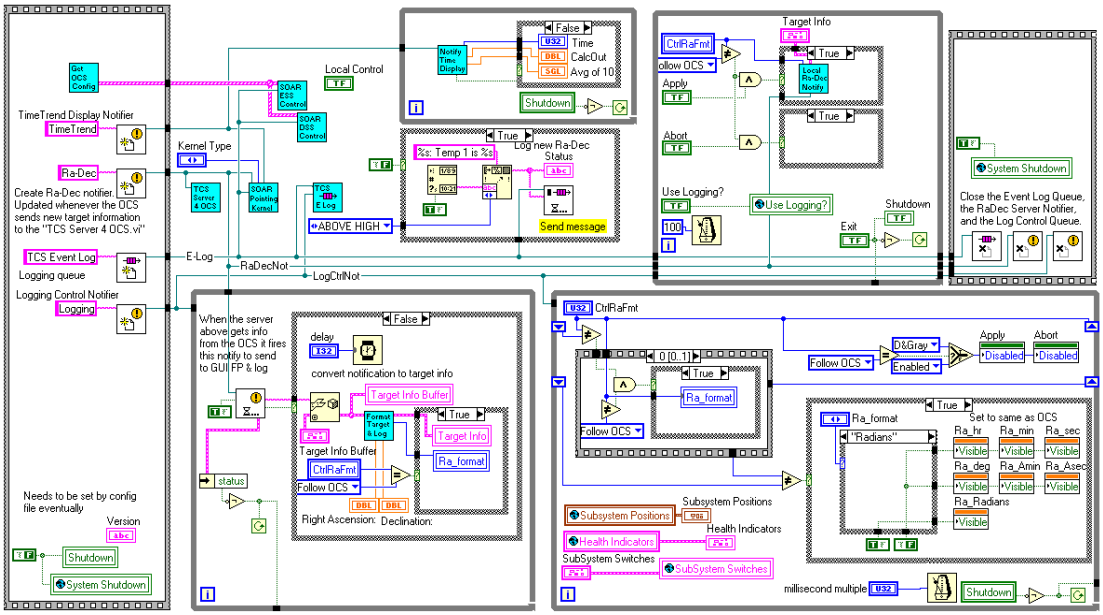

Figure 5.0 The TCS Module source code diagram: This shows four different parallel loops, all running concurrently.

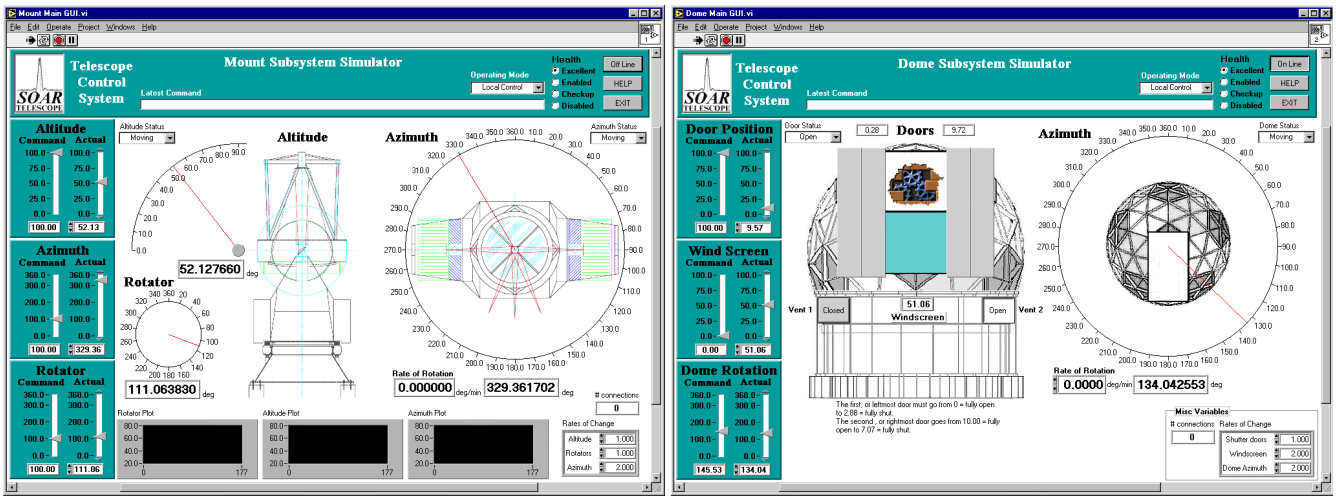

Figure 6.0 Mount Subsystem Simulator & Dome Subsystem Simulator GUIs. The positions of the Mount Altitude, Azimuth and Rotator are shown using a combination of customized LabVIEW controls, graphics from the SOAR project office and trend charts to show the motion history. GUI. The dome GUI at right is based on the previously developed Mount GUI. It uses many of the same controls and indicators. The diagram also reused much of the code from the Mount Simulator. This copying of previously developed code maintained connections from the diagram to the front panel controls.

2. REAL TIME LINUX

The SOAR TCS is partitioned into several subsystems, organized in terms of well defined functions. The subsystems are coordinated by what is called the TCS Application, which has the responsibility of making information readily available on time, for the system to work in a fluid manner. In this section we present the tests performed to ensure the correct execution of the processes.

The only time critical section of the TCS Application is the computation of the demanded mount position, and the transmission of the demand to the mount itself. This process is done every 50 msec, and even though it seems like a long time in terms of what one considers “real time”, it needs to be done on time, every time.

The TCS prototype under Windows, was showing signs of not fulfilling the demand, and the activity display was telling us that clock ticks were missing every so often. Instead of abandoning our LabVIEW environment, we looked for alternative platforms that can support “real time” tasks. Our search for alternatives leaded us to a variant of Linux known as RTLinux.

RTLinux (www.rtlinux.org) is a hard real-time variant of Linux, that provides the capabilities of running special real-time tasks and interrupt handlers on the same machine as standard Linux. These tasks and handlers execute when they need to execute no matter what Linux is doing. Typical response times are under 15 microseconds on RTLinux running on a generic x86.

RTLinux used in this prototype was version one. It provides very simple services to define real-time tasks. Simple FIFOs are implemented for transferring data between real-time processes and Linux processes. A simple fixed-priority scheduler is available. All in all, with very few money you can setup a very reliable real-time system for a variety of applications in the control area. As of today, version two is available with a POSIX compliant API, as well as several other independent distributions.

A test was designed to check whether this platform could be utilized with LabVIEW. LabVIEW is a multithreaded package, so in principle it should be possible to trigger the execution of a thread from the real-time kernel, and then play with priorities at the main Linux level to get the desired execution sequence.

The test is shown in fig. 7. The Linux processes consists of the three demand computation loops, fast, medium, slow, a dispatch process implemented using the *select* function pending on messages sent using the RTLinux FIFO mechanism, and

the LabVIEW processes. The dispatch program and the computation loops are compiled and make into a shared library. LabVIEW connects to Linux shared libraries by means of the Call Library Function.

The RTLinux side consists of an activation task and a flags control task. The activation task is awakened with a preprogrammed clock. The clock is running at 20hz for the fast loop, at 1 Hz for the medium loop and at 0.1 Hz for the slow loop. The flags are control structures that give an indication of the completion of the previous activation. If a loop hasn't finished when it's time to activate it again, an error condition is raised, so it's possible to check for any missing activation failure.

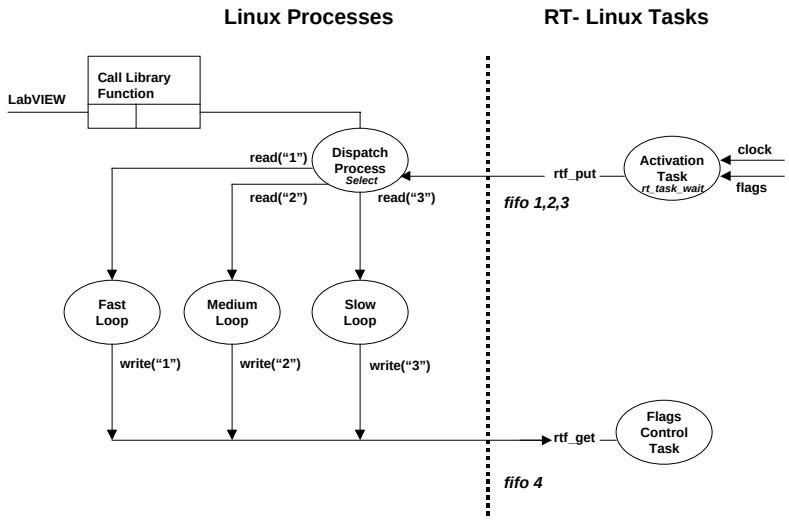

Figure 7.0 RT Linux - Linux – LabVIEW Functional Interfaces.

The sequence of events is as follows. LabVIEW calls the dispatch process and waits on the select function. The activation task is awakened by the clock, and the first thing it does is to check if the flag for the loop is set. If it's true then it resets the flag and send a message through a FIFO to the dispatch process. This process now triggers the selected loop which upon its completion, sends a set flag message through another FIFO to the flags control task. Then the dispatch process returns to LabVIEW with the updated information, and the entire sequence repeats again.

The test was run for many days and no missing activation was detected, giving a good indication of the robustness of the system. The next step was to modify the Windows version of the TCS application, to link with the new shared library in the Linux platform. This was a straightforward process as you can easily port VIs between different LabVIEW releases. The porting process was successful and it was decided to utilize RTLinux for the implementation of the TCS Application, as well as the Instruments control programs. WindowsNT will be utilized for the TCS GUIs, and the DataLogger in order to take advantage of the ActiveX capabilities of LabVIEW under that platform for display widgets and reporting using MS Office.

3. LABVIEW TUTORIAL

The software was developed in LabVIEW version 5.0.1f1 on Windows98/NT 4.0 machines. SkyMap Pro 5 was used as the star catalog. Graphics images were edited with PaintShopPro 5.0 and Visio Technical 5.0a. SLALIB and the Gemini Pointing Kernel were compiled with Visual C 5.0 under Windows and GNU C under Linux.

Since LabVIEW is a relatively new language to the astronomy community, a brief summary of its development environment, coding paradigm and built-in libraries is in order. LabVIEW is a compiled, general-purpose language with all the normal

control constructs and data types. LabVIEW differs from textual languages in several key ways: (1) The source code is entirely graphical, resembling a circuit schematic. (2) It has inherent parallelism of loops and control flow. (3) Execution order is data driven, i.e.; a node begins execution when data is available at all of its input controls, i.e., "dataflow". (4) The graphical user interface is built in to the routine and also implements part of the calling mechanism. (5) Although strongly typed, LabVIEW's built in functions are largely polymorphic. They auto-adapt to the wired data type. (6) Syntax errors are inherently minimized in LabVIEW.

A LabVIEW software routine is called a "Virtual Instrument" or "VI". Tool selection is via a "Tool Palette".

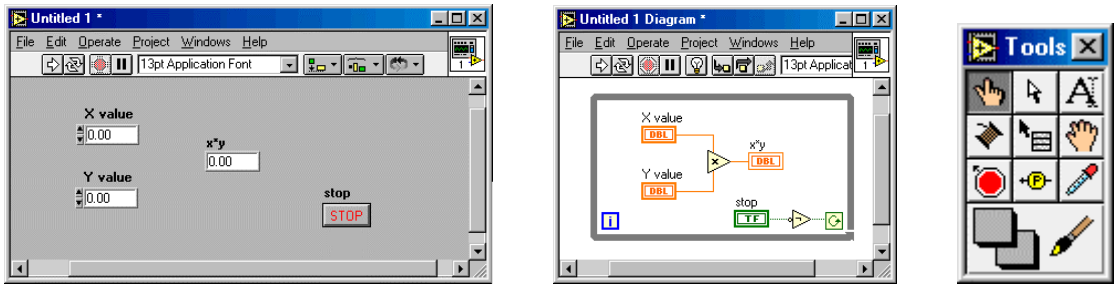

Figure 8 LabVIEW front panel and processing code diagrams The LabVIEW environment has menus of controls and functions that are available in palettes of icons. Drag and Drop is used to transfer items to the front panel and diagram.

Each VI is composed of three parts: the front panel (GUI), the code diagram, and the subVI connector. The VI shown above in Fig 5 uses the front panel (GUI) on the left to get X and Y from the operator and computes Z = X*Y in a loop in the diagram on the right until the STOP button is pressed, which exits the while loop. A VI may call any other VI as a "subVI". Collections of VIs may be stored in libraries. (*.llb files).

Main LabVIEW libraries for STP-1. In developing the STP-1 software heavy use was made of two built in libraries: **TCP/IP Communications** (subpalette to Communications) allows creating, closing connections, listeners, reads, writes and IP-string conversions. **Synchronization VIs** (subpalette to Advanced) has several submenus for creating notifiers, queues, semaphores, rendezvous and occurrences. These allow synchronization of VIs without polling loops. Essentially software interrupts with data passage attached to the interrupt. These libraries are available as submenus to the main palette. They are shown here along with the Control Structures menu.

LabVIEW Function Palette Includes 21 subpalettes of built-in nodes and VIs to perform various tasks. The last button on the bottom allows entering any subVI via a file dialog. The available palettes are: Control, Numeric, Boolean, Strings, Arrays, Clusters, Comparison, Time&Dialog, File, Instrument I/O, Drivers, DAQ, DSP, Math, Graphics&Sound, Communications, Application Control, Advanced, Reports, Tutorial, and User Libraries.

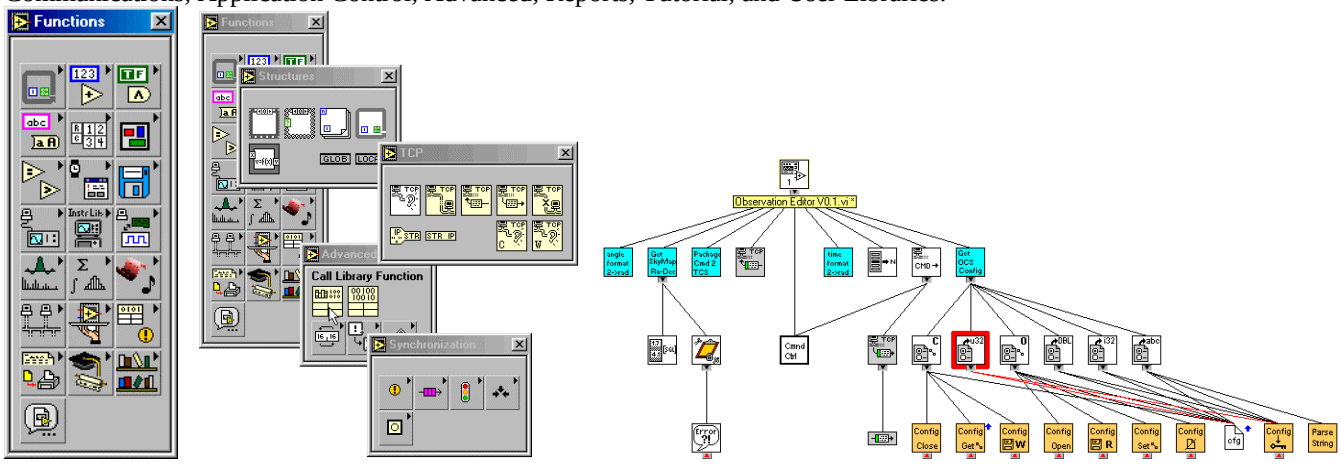

Figure 9 VI Menus, Submenus and Hierarchy: These were used extensively in the STP-1 design. All external communications were handled by TCP/IP. The TCS internal data passage was mainly by notifiers (one broadcasting to many), and Queues, (many sending to

one). Additionally, the "Call Library Function" in the Advanced palette was used to call the Gemini Kernel C code. A typical VI hierarchy (of the STP-1 OCS module) is shown on the right.

4. LABVIEW ADVANCED TOPICS

Although LabVIEW is a dataflow language and normally uses "wires" to connect nodes and pass data, there exists another method for indirectly calling VIs and manipulating VIs and controls either locally or remotely on another machine connected over the network. This method is called VI Server. VI Server can be either ActiveX or TCP/IP based. SOAR will use the TCP/IP method exclusively. The basic calling mechanism is illustrated below.[3]

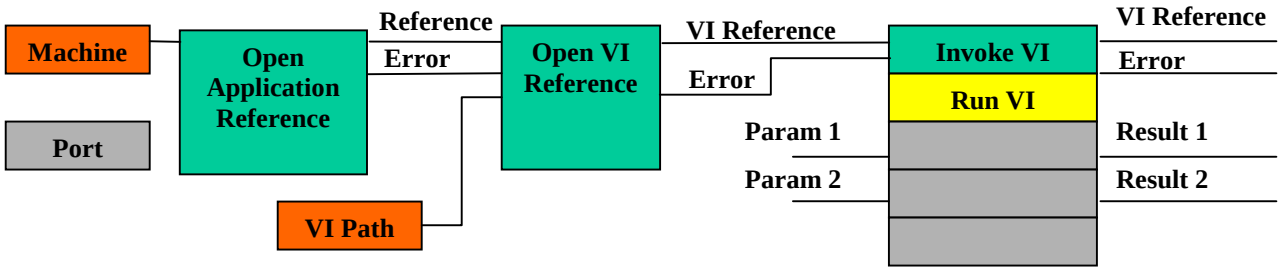

Figure 10 VI Server Mechanism: The VI Server allows control of a remote VI using a reference to that VI rather than the usual wires. All SOAR VI Server communications are handled by TCP/IP.

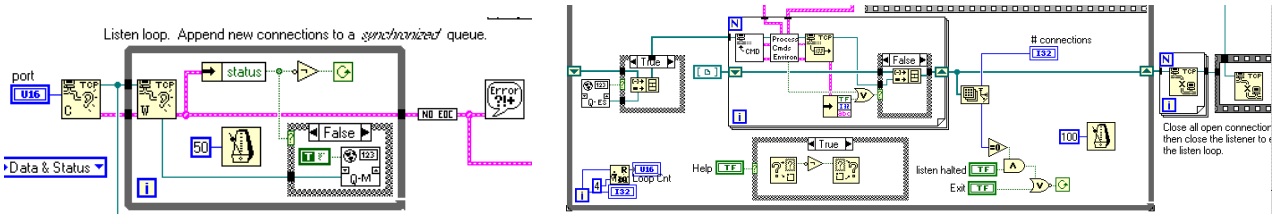

Figure 11 TCP/IP Listen/Process Mechanism: The STP-1 communications and some of the final TCS communications utilize a "Listen" (on left) code technique and "Process Message Queue" (on right) technique. These fragments were copied and reused throughout the STP-1. They will be modified and encapsulated into subVIs for use in the final TCS.

5. CURRENT TCS & DATALOGGER

The previous portion of the paper focused on the TCS prototypes. This section focuses on the current architecture of the overall TCS, DataLogger and subsystem interfaces as they will be implemented in the final versions. Figure 12 below illustrates the software architecture and functions in the modules of the TCS, DataLogger and TCS Subsystems. The TCS Application will be hosted on a Linux machine with a very minimal GUI. Normal interaction with the TCS Application shall be through the TCS Operator GUI which shall be hosted on a different machine running WindowsNT. The Instrument Computer(s) shall be based on Linux, and possibly the Calibration Wave Front Sensor (CWFS) Acquisition computer may be Linux. The Dome and Environmental Subsystem computers are currently scheduled to be WindowsNT machines. The Instrument Utility machine may be Linux or NT based. Although the architecture assigns functions to specific machines, almost any machine could be used to perform some of the functions of another machine should a machine become disabled. The platform independence of LabVIEW shall allow for rapid reconfiguration of the overall system in the event of casualties to an individual component.

This architecture shall minimize the effect of loss of a single machine, allowing the telescope to continue operations in a full-up or degraded mode upon loss of resources. Subsystems shall automatically reconnect communications after a casualty recovery. Actual control transfer shall be by manual operator action after verifying that the recovering module/subsystem is again functioning properly.

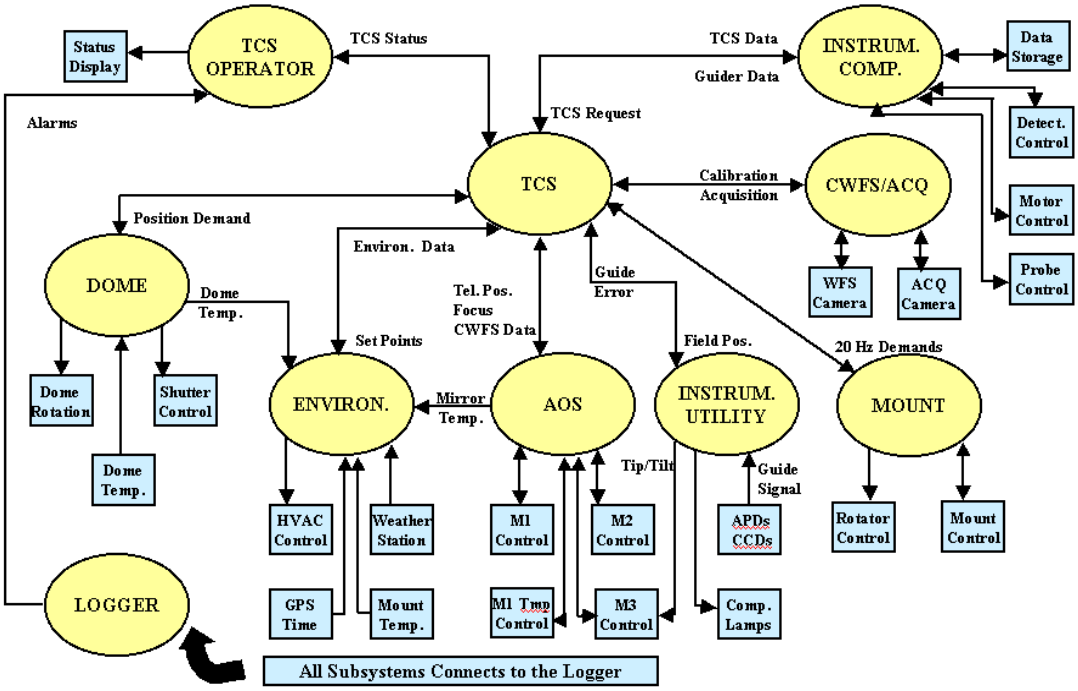

Figure 12 TCS Architecture: The current architecture for the final TCS[4].

The final TCS operators panel combines data display and control. 75 % of the area is devoted to displays that the operator should always have visible. The lower right corner has a multi-tab control area where most operator actions occur.

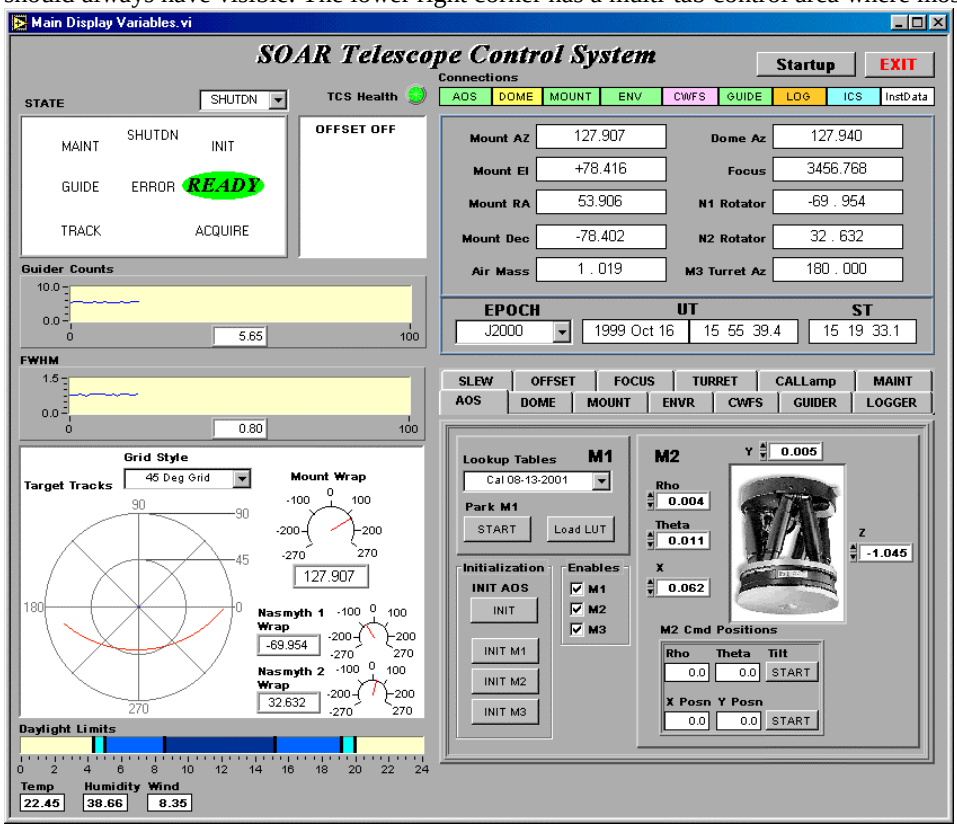

Figure 13 TCS Operators GUI: The current layout for the final TCS Operators GUI Uses a combination of custom and built-in LabVIEW controls.

DataLogger Architecture: The DataLogger implements several functions: (1) Captures Status, Alarms, Diagnostics data. (2) Tool based on command exchanges. (3) Variable Names Database, (4) Updates from each subsystem's database, (5) Selection of items to capture, (6) Query subsystems, (7) Information stored for DBMS access, (8) Status available system wide, (9) Subscription mechanism, (10) Data visualization.

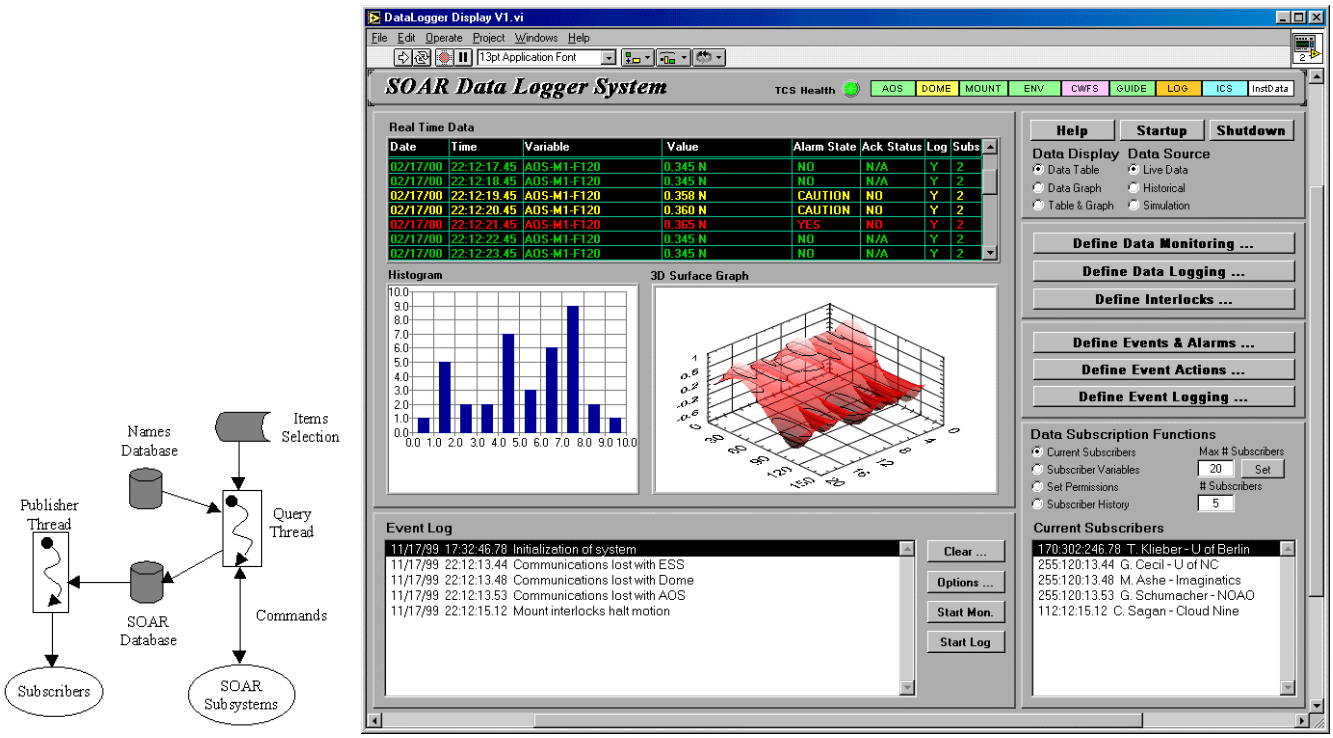

Figure 14 DataLogger Concept Diagram and DataLogger GUI: The data logging concept[3] gathers all variables exported from the TCS Subsystems into a Names Database. From this an Items Selected List is queried by a LabVIEW process using TCP/IP and VIServer communications. The collected data is logged to a three-part database using the Citadel database, an SQL database and a LabVIEW DataLogger database. Internal and external subscribers make a request to be sent updates of selected data items using the LabVIEW DataSockets interface. The GUI on the right displays an event log in the LL, a list of current data Subscriber information in the LR, controls and buttons to access setup dialogs in the UR, and an operator definable display area in the UL/middle.

6. LABVIEW CONNECTIVITY

LabVIEW is currently available in the following operating systems(OS): Windows 3.1/95/98/NT/2000, Linux/RTLinux (Redhat), Concurrent/Harris Nighthawk/PowerHawk, Mac, Sun Solaris, HPUX, LabVIEW/RT (Pharlap). Most of the functions are native LabVIEW "G" code and are portable from platform to platform. This has allowed the SOAR project to code various modules on the programmer's personnel choice of machine and OS and the rest of the team has shared the code on their platform by simply opening the code and resaving which performs an automatic recompile. This platform independence has reduced project costs by allowing reuse of available computer resources, rather than purchase of new ones for the project.

Graphical Object-Oriented Programming (GOOP): Careful encapsulation of key functions in "Virtual Instrument classes" is enabling reuse of communication libraries on all TCS and Subsystem platforms with the exception of the Mount subsystem which uses custom pre-written C code under the QNX operating system.

SOAR makes heavy use of two connectivity functions in LabVIEW: DataSockets and VIServer. VIServer is used mainly for control functions, both locally and on remote modules. DataSockets is used mainly for status data transfer. Both functions are built on TCP/IP.

DataSockets: Is a client-server data broadcast utility built into LabVIEW. It allows a server to be set up that broadcasts selected data to client subscribers. The SOAR DataLogger publishes data both internally and over the INTERNET using DataSockets. This allows remote subscribers to access live SOAR data using either a LabVIEW client, Visual C, Visual Basic or via a web page. Security: The DataSockets Configuration utility allows pre-defined data items or items can be created on-the-fly. The predefined items can be restricted as to which readers(clients) and writers(servers) have access to specific data items. SOAR uses this facility to insure that only the desired portions of the system have command broadcast capability. External clients access SOAR data through a firewall. DataSockets can work through a firewall using a free add-on. DataSockets has two methods for securing data transfer, as follows: (1) DataSocket Server Manager, which allows configuration which machines have permission to read, write, and create items on the DataSocket Server. (2) Running the DataSocket server on a machine outside the security firewall, while the data acquisition application remains inside the firewall. Buffered handshaking between the TCS DataLogger server and the various internal and external clients insures that all clients receive every update of data. The TCS interlock mechanism can use DataSocket in the multiple-writer mode to allow several TCS subsystems to update interlock variables which are then broadcast to the other subsystems. DataSocket bandwidth is mainly limited by network bandwidth. Benchmarks for 10BaseT are up to 320Kbytes/sec. SOAR's Fast Ethernet will achieve much higher data rates.

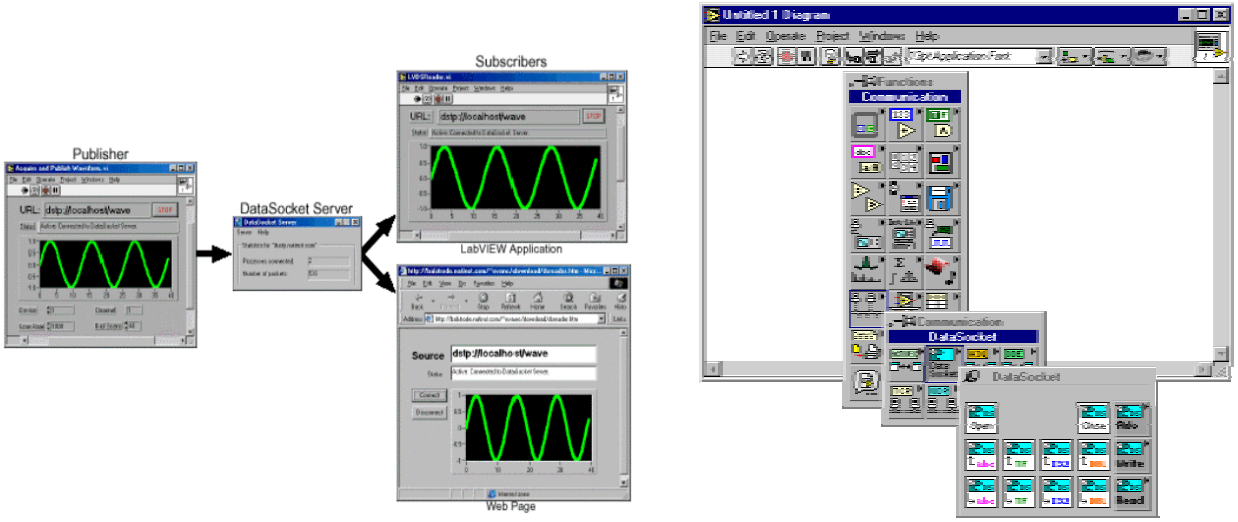

Figure 15 DataSocket Concept Diagram (left) and DataSocket VI Palette (right): The DataSocket allows publishers of data to send the data to a DataSocket Server, which may be on another machine, which then publishes data to remote LabVIEW clients and web pages. The DataSocket functions are accessed through the LabVIEW>>Functions>>Communications>>DataSocket subpalette.

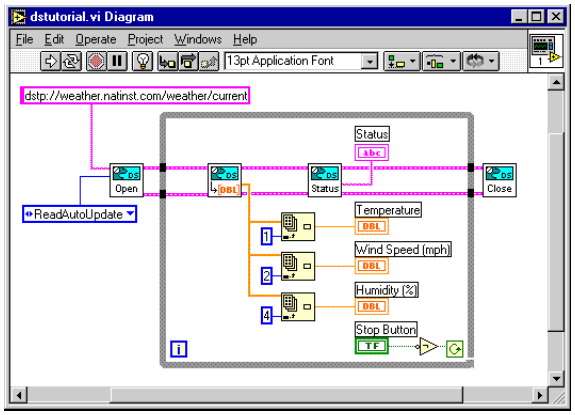

Figure 16 DataSocket Client Example Diagram: This example demonstrates the simple opening of a DataSocket reference, then looping on a DataSocket Read while updating display of three variables. The example could be used to view SOAR weather remotely by simply changing the URL location in the Open VI to point to the SOAR DataLogger server address.

VIServer: Is a method in LabVIEW to remotely load and manipulate LabVIEW VIs and global variables using references rather than wires. An example is shown in fig. X.

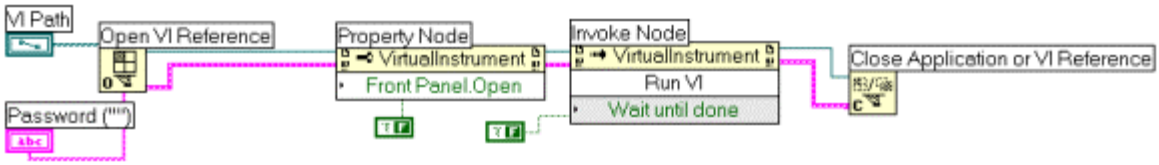

Figure 17 VIServer Example Diagram: The VIServer functionality is invoked by opening a named reference, then wiring in one or more property and/or invoke nodes. This is an object-oriented method that SOAR utilizes to create "plug-in" APIs.

The VIServer has setup and configuration utilities to provide for security.

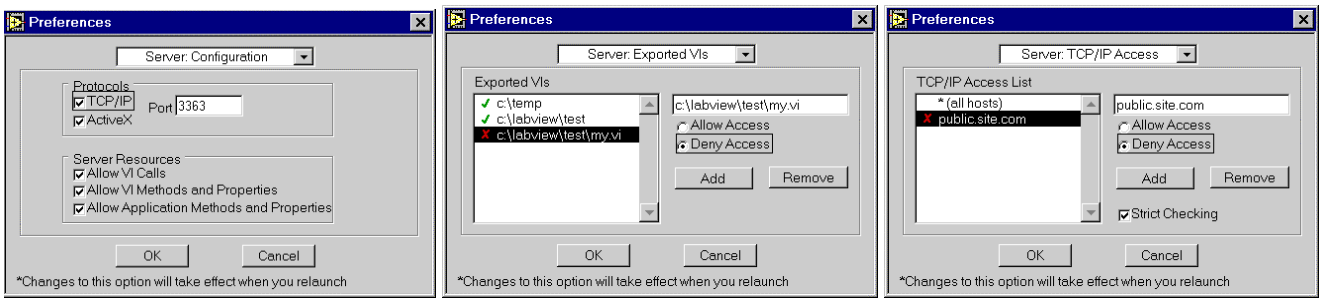

Figure 18 VIServer Preferences for Configuration, Exported VI List, and Allowed Access List GUIs: The VIServer in SOAR uses the setup and configuration panels to insure that only authorized access to command functions is allowed.

7. SUMMARY

The prototypes were completed on schedule, meeting all objectives of the design study. The performance of LabVIEW under various operating systems was studied. The merits of LabVIEW and it's capabilities with respect to developing an application suite of the complexity and robustness required for a Telescope Control System were well established, both by analysis and live demonstration. Of particular note was the robustness of a LabVIEW based TCP/IP communications layer, and the stability of LabVIEW – C code interfaces under Real-Time Linux.

The robustness of the SOAR communications layer was demonstrated during resolution of an unexpected problem in the STP-1 prototype. One of the laptops had a bad RJ-45 connector for the Ethernet connection. Debugging the problem lead to plugging and unplugging the connector over a dozen times. The demonstration was still running and each time the connector was unplugged the software smoothly dropped the client-server connection and re-established a new connection as soon as the physical connection was re-established. This occurred on the Mount simulator, which tracked to the last requested position, then paused. As soon as the connection was re-established it would slew to the new demand, then track as normal.

The Linux version of the TCS application ran flawlessly for several days in preliminary testing, never missing or delaying a single 20 Hz update.

Several issues related to software and OS selection were established:
(1) The most features are available under the WindowsNT OS. This is due mainly to ActiveX, which is not (currently) available under any other OS. This is slowly changing, as more of the ActiveX based functionality is being ported to native LabVIEW code. (2) The best stability and timing regularity was achieved under the Linux OS. (3) The LabVIEW environment utilizes the full bandwidth of Ethernet. In the prototypes this was 10BaseT. The final TCS will be implemented using Fast Ethernet in 100BaseT and Optic Fiber. (4) The Gemini kernel code runs with Linux better than NT. Further, the source code is more difficult to compile in the NT environment. Modifications were required, whereas the code compiled easily with no modifications under Linux. Accordingly, the TCS shall utilize both Linux (TCS Application and Instrument Control) and NT (DataLogger and TCS GUIs) depending on the requirements of the specific module(s). (5) The complex

sequences involved in startup, shutdown and state transitions of the telescope shall require a scripting language, in LabVIEW, which can automate the running of the TCS software and be modifiable by non-LabVIEW literate personnel. This will be provided by Imaginatics' "G'Script" scripting package.

Final specification of the TCS suite is completed and implementation of the TCS and DataLogger is underway and expected to be completed in late 2000 / early 2001.

1. T. Sebring, G. Cecil, G. Moretto, *The SOAR Telescope Project: A Four-Meter Telescope Focused on Image Quality*, SPIE 1998.
2. M. Ashe, The SOAR Telescope Control System Prototype-I in LabVIEW, NOAO Report, Feb 1999
3. G. Schumacher, *SOAR Concept Design Review*, June 1998
4. G. Schumacher, *SOAR Board Meeting Review*, Mar 2000