Enhancing International Space Station (ISS) Mission Control Center (MCC) Operations Using Tcl/Tk

Brian O'Hagan
NASA/Johnson Space Center
brian.ohagan-1@nasa.gov

Stephen K. Long, Sr.
United Space Alliance
stephen.k.long1@jsc.nasa.gov

Abstract

This paper will discuss the use of Tcl/Tk to enhance the abilities of flight controllers to control the International Space Station (ISS) from the Mission Control Center (MCC) at the Johnson Space Center. We will discuss why existing tools were not able to meet these needs as easily as Tcl/Tk. In addition, we will also discuss how we interfaced with the existing MCC infrastructure to receive ISS telemetry, find servers, register services, and send commands to ISS.

Extended Abstract

Existing MCC applications were developed for use on the X11 platform. These applications were typically coded in either C or C++. This resulted in the typical development limitations where changes where costly, time consuming, and difficult to implement. Several of these applications make use of Meta data files to allow easy definition of telemetry, commands, and computations though most only support one of these types. The application that came closest to supporting all of these functions was part of the Common Display Development Team (CDDT). Both the crew on ISS and flight controllers in the MCC use this application. This application is coded in C++ and combines the use of telemetry, computations, and commands on the same displays using XML data files. The main limitation of this application is its dependency on the ISS software configuration, which limits the delivery of updated data files to once or twice a year. Additionally, the application has limited graphical capabilities and a large backlog of deficiencies. We will discuss how we addressed these limitations using Tcl/Tk.

The Information Sharing Protocol (ISP) is the protocol used to pass telemetry, computations, and status data between workstations in the MCC. ISP uses the standard client server architecture and allows data to be multicast on a change-only basis unlike telemetry that downlinks all data all the time. ISP is coded in C and uses a library of APIs for accessing its functionality. We will discuss how we were able to interface Tcl/Tk applications with the ISP APIs to access telemetry data, register callbacks, commands for telemetry events, and provide for multiple server connectivity. We will also discuss telemetry data management, callback management, performance tuning, and publishing of data to other ISP clients.

Using the ISP interface, Tcl/Tk applications were developed for Caution and Warning management, external camera control, automated procedure management on ISS, and for displaying telemetry. These applications were developed in a shorter period of time, with fewer defects, and with easier to use graphical user interfaces while maintaining the existing rigorous Configuration Management and certification standards prior to their use during ISS operations.

The MCC is a distributed architecture that allows for user access throughout the MCC. Since users can log into any workstation to support a particular activity and ISP Servers run on each user's workstation, the typical approach of finding a server by connecting to an IP address cannot be used. Instead, the MCC uses Network Registration Services (NRS), which defines a list of service IDs with the associated workstation and port. Applications then use NRS to lookup a service ID to determine the workstation and port

to connect to. We will discuss how Tcl was interfaced to NRS to allow for registering service IDs and for finding services. This allowed the Tcl/Tk application Caution and Warning Status Tracker and Analyzer (C&W STAN), to provide server functionality using HTTP in a manner like web services for ISS Caution and Warning event management.

We will also discuss how we were able to interface Tcl with the ISS Command Server for sending commands to ISS and for receiving command related event updates. Like ISP, the Command System is coded in C and C++, and uses a library of APIs for accessing its functionality. Existing MCC applications were limited to only sending commands and receiving a command accepted or rejected response. They did not have the capability to verify a command executed properly with end item telemetry. This required operator intervention for all command verification steps. We plan to discuss how the commanding interface allowed us to automate routine operations, provide an easier interface for camera control, and an enhanced scripting capability. We will also discuss command data management, callback management, and performance tuning.

1. Introduction

This paper will discuss the use of Tcl/Tk to enhance the abilities of flight controllers to control the International Space Station (ISS) from the Mission Control Center (MCC) at the Johnson Space Center (JSC). JSC is the lead center for ISS operations for the National Aeronautics and Space Administration (NASA).

We will discuss the current MCC architecture and how we interfaced Tcl to the existing infrastructure's APIs. We will also discuss some of the design trade-offs for the extensions we created and how they meet our needs. Lastly we will discuss how these extensions are used by Tcl/Tk applications to solve real world problems for ISS flight control.

2. Architecture

The MCC uses a distributed architecture that is isolated from the outside world. It also provides access to telemetry and commanding from any MCC workstation. The MCC uses HP/Compaq Alpha workstations running Tru64 UNIX 4.0F.

The building is subdivided into Flight Control Rooms (FCR) and Multi-Purpose Support Rooms (MPSR). Typically Space Shuttle operations are performed from the White FCR and MPSR, ISS operations are performed from the Blue FCR and MPSR, and training operations are performed from the Red FCR and Blue MPSR. Testing and other activities can occur in any of these rooms or in one of the payload MPSRs.

To prevent users on one activity from interfering with another, the MCC uses a rudimentary form of Virtual Private Networks (VPN) called activity separation. Activity separation divides up the MCC resources based on the program, vehicle, certification mode, and flight identifier. This allows each activity to use recon, software, or data streams without interfering with other MCC activities. Users are only allowed to select from a list of currently valid configurations at login.

To separate data, storage areas are divided into certified and uncertified software and subdivided based on the flight ID. Only storage areas with the specified cert type and flight ID are mounted at login. This allows for the testing of new versions of software in uncertified activities without impacting ongoing operations in another certified activity.

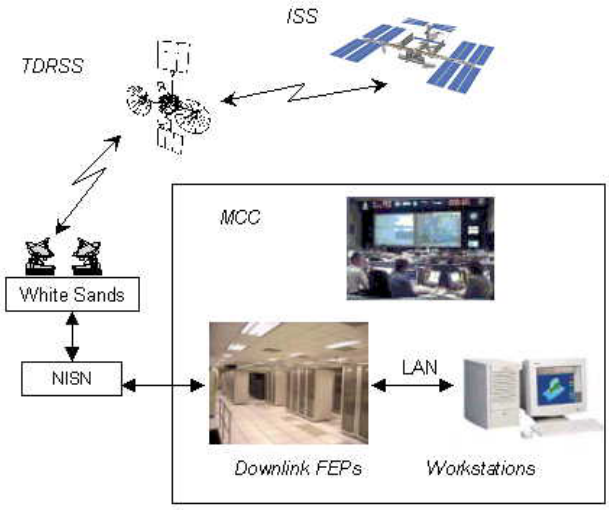

Figure 1: ISS Telemetry Path

Telemetry data is downlinked from the Shuttle or ISS through the Tracking and Data Relay Satellite System (TDRSS) to the White Sands Ground Station, then passed through the NASA Information Network to the Front End Processors (FEP) in the MCC. The FEP then

sends the telemetry data to the MCC LAN in a multi-cast data stream with a unique identifier based on the program, vehicle, activity type, flight identifier, and data type.

Since users can log into any workstation to support a particular activity, the servers and data streams can change based on the MCC activities. Network Registration Services (NRS) is used to find the appropriate data streams and servers for an activity. MCC applications query NRS to find the server providing the data stream or service ID matching the activity parameters and port for connectivity.

To receive data, a Data Acquisition server is started on the user's workstation. The Data Acquisition server uses NRS to find the specified service ID based on the activity parameters, then connects to the specified data steam. Applications can then either connect to the Data Acquisition Server (also found via NRS) to receive all of the data via a Point-to-Point protocol, or connect to an Information Sharing Protocol (ISP) Server. The ISP Servers connect to the Data Acquisition Server but, ISP clients only receive changes in the value or status for parameters they have subscribed to. This greatly reduces the processing needed by client applications. To put this in perspective, ISS has 95,000 parameters being downlinked in a 192Kbps data stream which updates all parameters every 10 seconds. Some parameters update at 1 Hz, but most update at 0.1 Hz (every 10 seconds). The majority of parameters do not change on a regular basis, which would result in a lot of wasted processing. ISP also allows for applications to publish computations for other ISP clients to subscribe to.

Since the downlink telemetry depends on the ISS S-Band communications system, the scheduling of TDRS satellites, and blockage between the two, we may have a Loss of Signal (LOS) and Acquisition of Signal (AOS) several times per orbit (~90 minutes). For each AOS and LOS, all parameters toggle from "static" to "nominal" status and vice versa. This results in bursts of ISP updates at each AOS and LOS.

3. NRS

NRS is an application that runs on every MCC workstation. It provides a C level API for applications to send registration, deregistration, and search requests and receive the results. Whenever a new service is added to the MCC, its ID is sent to the NRS application on each MCC workstation. Likewise, when an NRS service ID is no longer used, it is removed by each NRS application. Since the list of services is resident on each workstation, queries return a result immediately.

The NRS extension is a complied C package, which wraps the NRS APIs to provide a Tcl interface. When loaded into a Tcl interpreter, it creates the "nrs::nrs" command. This command provides sub-commands to access the NRS APIs. The extension uses the Tcl object APIs and returns query results as Tcl list objects. The extension handles cleaning up for itself if the nrs::nrs command is deleted. The Tcl event loop is not needed for the extension to work.

Command	Returns
nrs::nrs assign_port service_id	Assigned port number
nrs::nrs deregister service_id	Boolean status
nrs::nrs help	Extension help
nrs::nrs host_search host_id	List of Service IDs on Host
nrs::nrs port_search service_id	List of Host IDs and port numbers
nrs::nrs register service_id	Boolean status
nrs::nrs register_port service_id port	Boolean status
nrs::nrs search service_id	List of Hosts for service
nrs::nrs service_search pattern	List of Host, Service ID, and ports matching pattern

Table 1: NRS Extension Command Usage Summary

Register/Deregister Functions

The NRS extension allows Tcl applications to register a NRS Service ID and optionally have NRS assign a socket port. The extension also allows a Tcl application to deregister a Service ID. Using these capabilities, a Tcl application can act as a server for MCC services.

Search Functions

The NRS extension also provides the capability for Tcl applications to query workstations or servers providing a specified Service ID or for querying all services available on a particular workstation. There is a limited wildcard searching capability.

4. ISP

Like the NRS extension, the ISP extension wraps the C level ISP APIs, to provide an interface for Tcl applications. Unlike NRS, ISP data updates are event driven and require the Tcl event loop to run. When loaded into an interpreter, the extension creates the “isp::isp” command. This command provides sub-commands to allow Tcl applications to access the ISP extension functions.

Contexts

The ISP extension supports establishing a connection to an ISP Server by creating a new command for each ISP Server connection (a.k.a. context). To establish a connection, the user application calls the initialize function with the server to connect to, an application identifier. The extension returns the new command for that context, like Tk does for new widgets. Using this new command, the application can then connect or disconnect to the ISP Server, enable or disable the ISP connection (turn on/off the flow of data), subscribe or unsubscribe to telemetry parameters, register callback commands for value/status updates, get the value or status of a parameter, publish or unpublish parameters to the ISP Server, set parameter filters in the ISP Server, retrieve context status, query the ISP Server for status, or register callbacks for ISP packets.

Like any other ISP client, the user application needs to initialize the connection, so the ISP API knows which Server to find via NRS. The connect sub-command does the connection to the ISP Server. The subscribe sub-command tells the ISP Server which parameters to send value and status updates for.

Data Updates

ISP sends the current value and status when an application first subscribes to a parameter and whenever that parameter changes. This blends well with the Tcl way, since the event loop can be used to handle socket polling. Data updates are sent in value, message (string), or status packets. The extension then stores all received data in C structures using a Tcl hash table with the parameter ID as the index.

This presents a bit of a problem. The get sub-command allows the user application to get the current value or status at any time. However the application doesn’t know when the value or status has changed. Also some telemetry parameters can update at 30 Hz.

To ease the burden of constantly checking for changes, the extension allows the application to register callback commands to evaluate whenever the value, sample number, status type, status color, status symbol, or validity of a parameter changes. This registration allows for an unlimited number of callback commands by using a Tcl list of lists. ISP sends all value and status updates in the first third of a data update cycle (roughly 1 second). The approximate time between events is 20 microseconds. This allows for computations to receive their input values then publish a new value within the same data update cycle. The downside to this approach is if the registered callback commands take too long to evaluate, the receive buffer may overflow and result in the lost of data. Using this approach, there is also a noticeable jump in the CPU usage for parameter processing and there could be several display updates.

The alternative approach is to store all value and status updates until the end of the data update cycle, and then evaluate all callback commands. This allows for an efficient use of the CPU since the data updates are done before computation and then display updates are performed at once. However this is not as desirable for

computations since anything published after the end of the data update cycle is not sent to other clients until the next update cycle. Therefore, published parameters are no longer time homogenous with their input values.

Both approaches have their advantages and disadvantages. For computations, the former approach is optimal if the computations are not very complex, but for display only applications, the latter approach is optimal. Testing has shown it would be too complex to allow for a combination of both approaches at the parameter level. The solution was to set the update mode at the context level during the initialization phase. Applications that need both capabilities could then create two ISP Server connections with one set to perform the callback evaluations immediately and the other to perform the callback evaluations at the end of the data cycle. This allowed for the best of both worlds without a large increase in complexity.

Data Values

ISP sends all symbols as either a double or as a string with length field. Since most Tk widgets and the Tcl format command do not like floating point values when performing integer conversions, this presented a problem. The choice was to either force the user callback command to do double to integer conversion or include it in the extension. The other problem is that unless a parameter is looked up in the recon to determine its true type, it would be impossible to know which type to convert to. To solve this problem, the ISP extension converts any value without a fractional component to either an integer or long integer, if it is within the value range for those types. Ideally it would be nice if the format command could do this for us. The converted value or string is then converted into a Tcl object and stored in the structure.

The second issue was with ISP time stamps. An associated time stamp accompanies each value or status update. This time has a dual use of either Mission Elapsed Time (MET) for the Shuttle or the time since the beginning of the year. This time is in floating point hours. In order to reduce the processing needed to view time fields, the ISP extension converts the time into seconds such that it can be passed to the clock command without further processing. Unlike the value updates, this is only performed when the get command is used.

Other Features

To reduce the processing needed by the Tcl applications and to increase the extension performance we also made the following enhancements: value and status details (not the data value) are only converted to Tcl objects when a callback or get operation is performed. The results of all commands are returned as Tcl objects. A fast evaluation procedure was created to handle callback evals and call the C procedure directly to eliminate the conversion to byte code. Rather than using append functions when returning a list of values to the Tcl application we preallocate an array of object pointers, create the objects, then convert it to a list object.

The ISP extension also implements the ISP heartbeat function such that if the ISP Server or the client application does not respond for 10 seconds, an automatic disconnect will occur. The extension will automatically attempt to reconnect every 10 seconds using the Tcl timer handler. When the connection is re-established, all subscribed, published, or filtered parameters will be resent to the ISP Server.

When a parameter is unsubscribed, the interface sends a “Missing” status to all existing callbacks then deletes the hash table entry, structure, and all associated callbacks from the internal storage area to insure old and possibly wrong data is not used. Attempts to get an unsubscribed parameter, or if the subscribe was rejected, will return an error.

Overall the extension provides a very efficient interface for processing telemetry data. During testing, when we subscribed to 95,000 symbols, the CPU load for handling data updates was negligible with a virtual memory size of 75 MB for cyclic evaluations and 42 MB for immediate evaluations. It should be noted, that this tool did not do display updates.

Command	Returns
isp::isp help	Extension help
isp::isp init ?args …?	Initialize context
contextName callback type command ?args ...?	Add or remove callback command for ISP packet
contextName cget ?args …?	Get context configuration details
contextName connect	Connect to the ISP Server
contextName deregister type symbol command ?args ...?	Deregister the callback command for data or status update.
contextName destroy	Destroy the context and Tcl command
contextName enable	Enable the ISP Server connection
contextName disable	Disable the ISP Server connection
contextName disconnect	Disconnect from the ISP Server
contextName filter ?symbol? ?args?	Set filter(s) for parameter in ISP Server
contextName get symbol ?args?	Get parameter value or status details
contextName publish ?symbol? ?args?	Enable publication of parameter
contextName query flag	Query ISP Server for status
contextName recycle	Disconnect then reconnect to the ISP Server
contextName refresh	Request the ISP Server resend the value and status for all subscribed parameters
contextName register type symbol ?command? ?args ...?	Register the callback command for data or status update.
contextName set symbol ?args?	Publish a value and/or status for parameter
contextName subscribe ?symbol? ?-mask value? ?symbol -mask value ...?	Subscribe to a parameter
contextName unpublish symbol	Disable publication of parameter
contextName unsubscribe symbol ?symbol ...?	Unsubscribe to a parameter

Table 2: ISP Extension Command Usage Summary

5. Testing

The Software Management Plan (SMP) defines our testing process. It defines the required guidelines for the development, testing, and certification for all MCC applications. In addition to this, any application which affects processed telemetry or commands that are uplinked to ISS such that a user cannot verify the calculation, also falls under the Critical Processor Certification guidelines. These documents were developed from the perspective of compiled code so we have had to make some minor changes for Tcl applications. For the testing, we perform verification (a.k.a. unit testing), validation (a.k.a. integrated testing), and certification testing (in as flight-like an environment as possible).

To test the capabilities provided by the NRS and ISP extensions, we developed a variety of test tools. Although the purpose was to exercise the functionality of the extensions, they proved flexible enough to provide a base for subsequent applications.

The first test tool is a GUI application that provides the user with complete access to all of the ISP extension functionality. It displays a set of ISS telemetry parameters called Program Unique Identifiers (PUI) along with the current value and status and also basic ISP Server status. It also has the capability to show a history of functions performed by the user. General functionality is available from the menubar and parameter specific functions are available from a pop-up menu. This tool made testing much easier than previous ISP test tools which required a combination of command line tools and difficult to use GUIs. The tool proved so useful, it uncovered 10 previously unknown problems with ISP APIs and the ISP Servers and will be used to test subsequent releases of the ISP system.

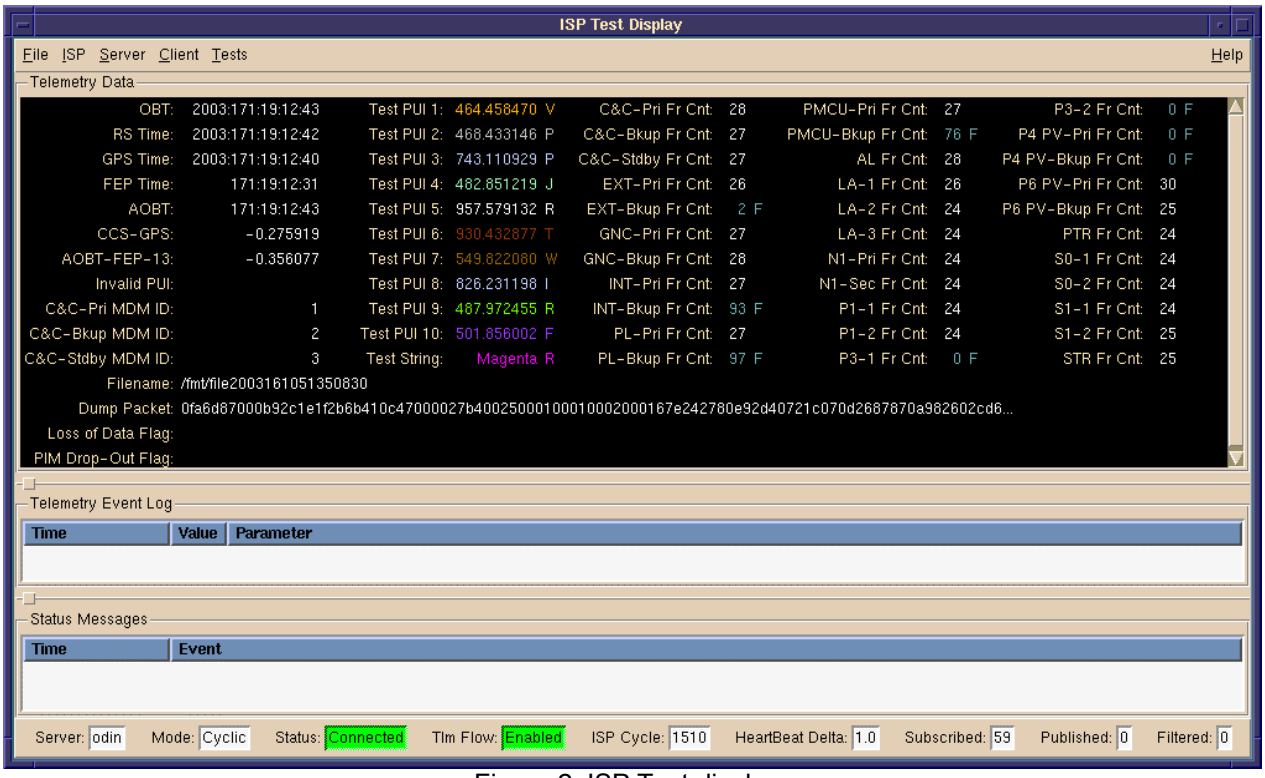

Figure 2: ISP Test display

Since ISS flight control is a 24/7 operation with many applications running for weeks or months between restarts, we also instituted stress testing. These tests were run continuously to exercise the init/destroy, connect/disconnect, subscribe/unsubscribe, publish/set/unpublish, register/deregister, and steady-state operations for at least 24 hours yielding anywhere from 40,000 to 60,000 cycles. This was very useful in finding memory leaks and other unstable areas of code. This testing also provided a baseline for CPU performance and memory usage under off-nominal conditions.

The second application was created to exercise all of the valid and invalid permutations allowed by the ISP extension. Since Tcl is a scripting language, we decided to use its scripting capabilities to automate testing and to split the original application into several applications. The first one would step through test cases of each major function of the interface, another one would subscribe to all 95000 valid parameters and report on the delay between data cycles, and the last one exercised connecting and publishing data to one ISP Server and receiving the same data through another ISP Server.

Likewise a similar test application was created for the NRS extension. It steps through each support function and passes both valid and invalid data to the register, deregister, and search functions. This tool also doubles as the stress tester by continuously running through all test cases for at least 24 hours.

6. Applications

Existing MCC applications were developed for use on the X11 platform. These applications were typically coded in either C or C++. This resulted in the typical development limitations where changes were costly, time consuming, and difficult to implement. Several of these applications make use of Meta data files to allow easy definition of telemetry, commands, and computations though most only support one of these types. The application that came closest to supporting all of these functions was part of the Common Display Development Team (CDDT). Both the crew on ISS and flight controllers in the MCC use this application. This application is coded in C++ and combines the use of telemetry, computations, and commands on the same displays using XML data files. The main limitation of this application is its

dependency on the ISS software configuration, which limits the delivery of updated data files to once or twice a year. Additionally, the application has limited graphical capabilities and a large backlog of deficiencies.

With the capabilities provided by the NRS and ISP extensions, we were able to develop several applications to address the above limitations and improve ISS operations. The current focus is on filling the gaps in the existing capabilities rather than duplicating an existing display.

The Caution and Warning (C&W) Management application is used to monitor changes in the C&W system on ISS and log them to a ground database. This consists of tracking when events go into alarm, return to normal, and their current annunciation state. Tcl/Tk was chosen since it allowed us to use a HTTP/XML client/server interface with a quick turn around for changes and easy to use displays. Previously this was tracked using a spreadsheet posted to a web page. We frequently had issues with the spreadsheet getting out of sync with the onboard system.

The biggest issue with development of the C&W management application is the idiosyncrasies of telemetry updates. C&W uses a queue of the last 20 C&W events from the primary computer on the US Segment of ISS and several counter and pointer parameters to determine when an event has occurred. Each event consists of 7 parameters. However, since the parameters are in different telemetry frames and packets may be dropped or corrupted (packets are not checksummed) along the way in addition to the differences in the data update rate, it is very difficult to know precisely when or what event has occurred. This has led to additional logic to correlate event and counter updates and to filter out ratty data.

The External Camera Control application incorporates the Tcl/Tk ISP interface and the Tcl/Tk Command Server interface to support the Communications and Tracking Officers with managing the External cameras onboard ISS. The implementation of this application uses both a Tcl/Tk GUI interface and the Tcl/Tk keyboard bindings to rapidly uplink camera control commands for smother operations. In addition, this application offers the capability of entering a Pan and/or Tilt angle for the camera to transverse Pan/Tilt positioning.

The previous method of controlling the ISS external cameras was with the CDDT displays. The CDDT display required the flight controller to select a directional button to build the command, then select a second “Are you sure” button to uplink the command, and finally the controller is required to wait for a response from the onboard system that the command was accepted. This approach added a systematically latency that hampered the flight controllers ability to control the camera.

The Automated Procedure Viewer (APV) application was developed by CDDT. This application was designed to manage automated procedures that have been converted from manual procedures. These scripts are written in User Interface Language (UIL) that was developed by Charles Stark Draper Laboratory. When the APV application was ported into the MCC, the number of deficiencies rendered it unusable. At this point the flight control team decided to develop their own application, with the look and feel of APV, using Tcl/Tk. Tcl/Tk was chosen for its rapid development capabilities, and large library of widgets.

Although there are existing displays to view telemetry, they are all limited in their ability to be configured. To address this we have also developed several applications that can display telemetry or perform computations using reconfiguration files or other dynamic input such that the application can use new data when it’s available rather than forcing the users to update the display or configuration files.

7. Future Directions

We are currently working on a commanding interface similar to the ISP and NRS extensions. These projects were delayed due to several issues in insuring the applications receive all command data updates and providing a way to associate data or requests sent to the Command Server with responses received by the application. Unlike the ISP interface, they won’t store all received data since the command database can be up to 50 MB in size depending on the number of commands in use and the system load. Also we plan to institute additional checks to insure commands are accurately uplinked.

We are also evaluating several other possible uses of the ISP, NRS, and new command interfaces.

8. Conclusions

Overall, we have not had any insurmountable problems in interfacing Tcl/Tk applications into ISS flight control operations. Analysis of the pros and cons of when to use Tcl/Tk has led to a new generation of tools that make flight control easier and more productive. Compared to previous applications, these applications were developed in a shorter period of time, with fewer defects, and with easier to use GUIs. We also did this while maintaining the existing rigorous Configuration Management and certification standards prior to their use during ISS operations.