Tim Liu **February 18, 1997**

NIRSPEC Software Design Note 11.00
Client-Server Architecture Design

1 Introduction

Remote observing is one of the functions to be provided by the NIRSPEC software according to the requirements specification. For the existing Keck instrument, the remote observing is carried out such that the observer at Weimea logs into the summit computer and runs the X-window-based instrument control GUI on the observer's local X display. Because the instrument software is not client-server (C/S) based, only single-point control is allowed. For NIRSPEC, however, we want to develop a fully C/S based architecture under which multiple user-interface clients can be executed concurrently and remotely via TCP/IP networks. With this scenario, users at different places will see the same GUI on their screens. For example, if one user selects a new filter position, which updates the GUI screen, the other clients will see the same graphical and textual changes on their screens. Figure 1 illustrates the multi-user control concept for the NIRSPEC server. The user interface client, which includes the Keck keyword library, communicates with the server using remote procedure calls (RPC).

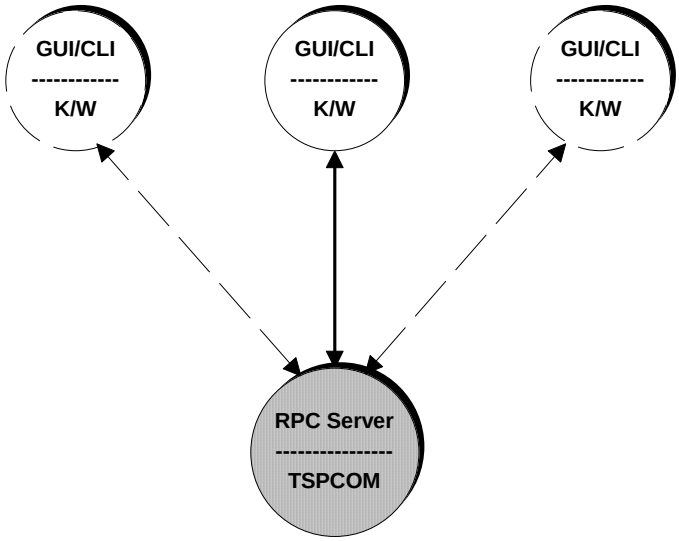

Figure 1 Multi-user control concept

2 Requirements

To provide a real multi-user remote observing environment, the NIRSPEC software must be designed and implemented around a client-server architecture. The most basic requirements for this software are:

- provide two servers: a real instrument server that connects to the underlying transputer system and a simulation server that runs without hardware
- all user-interface frond-ends like GUI, CLI, and quick-look are treated as clients
- client-server communications are via TCP/IP networks and implemented using industry standard message system

More specifically, the instrument server should provide the following functions:

- multi-client connections and disconnections
- server-to-client "broadcast"
- server security and access control
- access to the underlying transputer system
- image data I/O and storage

The simulation server should have similar features except that it does not need to talk to the transputer system.

In a C/S environment, there's an issue of whether to do a job locally or in the server. For performance consideration, we prefer a "thin" client, i.e., the server should do a job as much as possible.

3 Structure

As suggested by its name, a C/S system consists of two major components, the client and the server. The server listens to client requests and performs various system service functions such as controlling the instrument, while the client software provides the user-interface functionality. The server program resides in a single computer, but the client can run from a different site.

A communication mechanism links the server and clients together. The communication system in a C/S system is often referred to as "middleware". We have chosen RPC (remote procedure call) as our IPC (inter-process communication) mechanism. The major advantages of using RPC include: it is one of the standard IPCs in UNIX network programming; it has been widely used and has plenty of programming books available. It is very likely to remain supported by the industry in foreseeable future. In UNIX, RPC comes with two different flavors: HP's NCS (Network Computing System) RPC and Sun's ONC (Open Network Computing) RPC. The version we are using is ONC RPC as bundled with Solaris.

Figure 2 shows the NIRSPEC client-server architecture using RPC. The client side includes the GUI and CLI (the quick-look tool is also part of the client, but not shown here). As required by

the Keck Software Coordination Committee (SCC), a keyword library layer interfaces between the applications and the underlying hardware. Under this keyword-style control flow approach, user inputs for application control are sent to the server via *keywords* and application data (status parameters, not image data) are also returned to the user interface in *keywords*. Therefore, the KTL (Keck Task Library) layer and NIRSPEC keyword library are part of the client. The keyword library interfaces directly with the RPC layer via RPC API routines. The heart of the RPC server is an asynchronous event loop which handles communications and makes remote procedures calls. Most of the remote procedures in the server program fall into one of the three categories: routines for communication with the transputer system, routines for handling image data and file I/O, and routines for broadcasting messages.

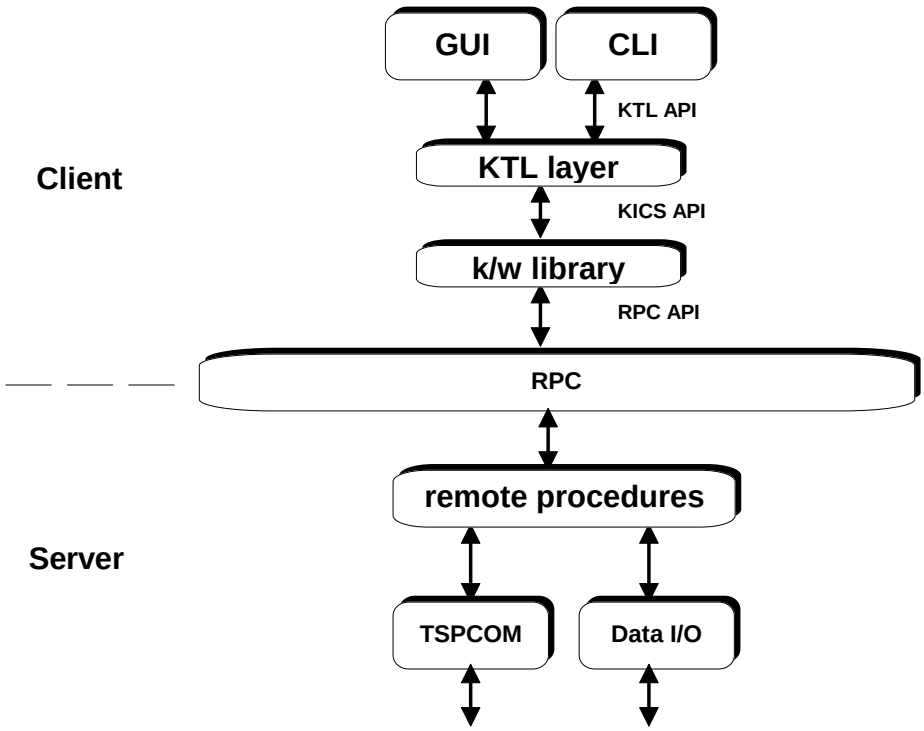

Figure 2 NIRSPEC client-server structure

The following example helps understand how the NIRSPEC C/S architecture works. When the user clicks the "go" button on the GUI, a KTL routine will write the *go* keyword to the NIRSPEC server via RPC. When the server receives this keyword, it first calls a remote procedure to pass it to the transputer system to initiate the exposure and then broadcasts this keyword to all the clients to start each individual's exposure timer. When the integration is finished, the transputer sends the *framerdy* keyword to the server and the server in turn broadcasts it to all the clients to stop their exposure timer count-down. The server then reads the pixel data from the root transputer and writes the data into disk in FITS format.

4 Specific Issues

This section addresses several specific and important issues involved in the design of the NIRSPEC server program.

4.1 Security and Access Control

Given the distribution nature of the C/S architecture, it is important to implement a reliable security scheme in the design. The security check should be solely handled by the server. It maintains an authorization file which lists clients that have permission to access the server. When a client tries to establish a connection to the server, the server will check its login and host id and look up the authorization file to decide whether to accept or reject the connection request.

The NIRSPEC instrument is controlled via *keywords* and some of the keywords are more important than others in terms of security requirements. For example, the keywords used to control DCS, motors, and exposure should be executed by users at the instrument computer console only. Under this consideration, all keywords are divided into two categories: special keywords such as exposure, abort, motor movement, and DCS control and normal keywords which perform less sensitive functions such as observing parameter setup. The keyword classification is defined in the keyword table maintained by the server. Similarly, a client can be assigned a status of either "principle user" or "secondary user". By default, the principle user is the one signed on from the control. Only one principle user is allowed at any time. All remote logins are secondary users. The principle user can transfer his/her special status to a secondary user. For example, an observer may carry out his/her observing from Weimea and therefore needs all the control to the instrument. Combining the keyword and user classification schemes, we can see that the principle user has access to all the keywords, while a secondary user can only access the normal type keywords. The client access rights are also defined in the authorization file maintained by the server.

A sample authorization file is listed below to illustrate the user access control scheme discussed above:

```
# NIRSPEC authorization file
# user:host:access
nirspec:crab.astro.ucla.edu:1
liu:crab.astro.ucla.edu:2
nirspec:jupiter.astro.ucla.edu:2
nirspec:kapolo.keck.hawaii.edu:2
```

The first two fields of each line list the login and host name of a permitted client, and the last field assigns the client's access right where "1" is the principle user and "2" means a secondary user.

4.2 Data I/O Scheme

In a C/S architecture, information storage should be centralized as much as possible. In the case of the NIRSPEC data I/O design, image read/write/storage, FITS header information, and file management should be completely handled by the server. A client should have no knowledge of them. Figure 3 illustrates this idea. When an integration is finished, the transputer system sends a frame ready signal to the server and the server relays it to all the clients. The TSP (Transtech SCSI Processor) routines then send the pixel frame to the server and the server file I/O routines convert the frame into a FITS file with appropriate FITS header keyword values and file name. For speed, the initial frame data buffering and the file conversion process are carried out in the memory-based file system **`/tmp`**. Converted FITS files are copied to the disk for final storage.

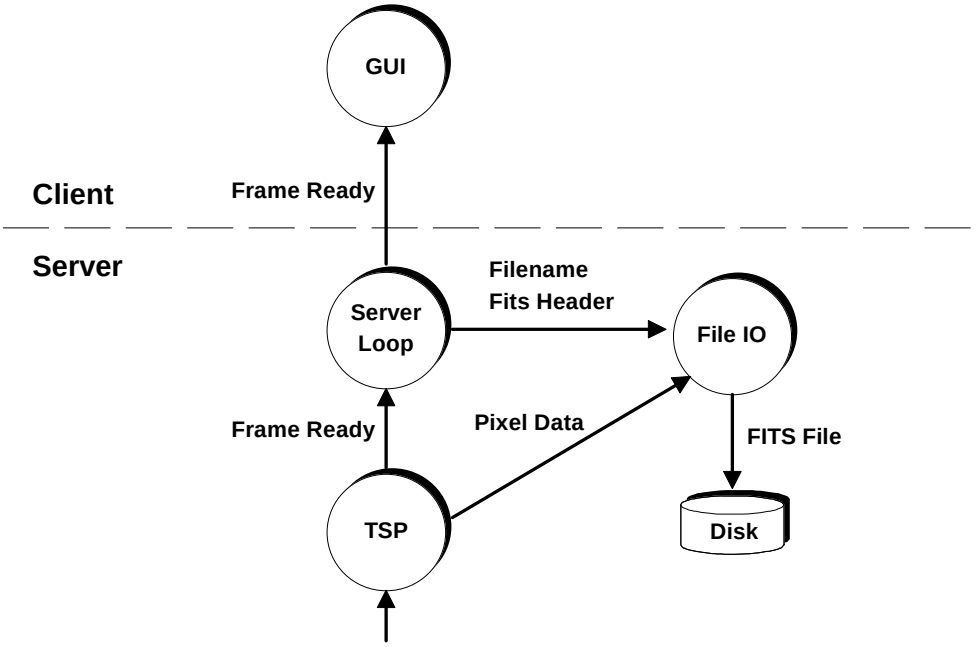

Figure 3 Data I/O scheme

4.3 Remote Data Transfer

An arriving image frame is stored in the server computer. If the client program is not running on the same machine as the server, a copy of the frame needs to be transferred to the client's computer for quick-look. This transfer process should obviously be automated. Although an RPC-based data transfer mechanism can be built into our C/S architecture, it is much easier and probably more efficient to simply use a standard file transfer tool like ftp. The implementation of the remote data transfer using ftp is straightforward: when a client that is not on the server computer receives a frame ready keyword from the server, it will invoke an automatic ftp shell program to get the FITS file from the server's data directory and then notify the quick-look to display. Because of the large data size and therefore considerable transfer time, the ftp program should run as a child process so as

not to block the main user interface process. The client should also be able to disable the automatic data transfer if the transfer time becomes intolerantly long. In addition, there should be some synchronization mechanism on both the client and server sides to make sure right frames are transferred in right sequence.

To shorten the data transfer time, an image file should be compressed. It is shown that the size of a NIRSPEC simulation frame (in 32-bit integer FITS format) can be reduced by half using the standard UNIX "compress" utility. Compression tools using more advanced algorithms may achieve even better results. Also, the time taken by the compression process can be reduced substantially with more powerful computers. However, no matter what compression tools we use, it will remain a challenge to transfer a NIRSPEC frame (4 MB uncompressed) within a tolerable time period, say 1 minute or so, via the congested Internet, unless we use a dedicated high-speed line.

5 Development

To beginners, developing a RPC program can be a complicated process. Even a simple RPC application requires a good understanding of the RPC internal mechanism and substantial amount of coding and testing. To ease the programming efforts, ONC RPC provides a protocol compiler called **`rpcgen`** which automatically produces client and server stubs that use lower-level RPC calls and handles data representation. All you need to do is to write a protocol definition file and use this utility to generate the required source code (you still have to develop remote procedure routines by yourself). However, **`rpcgen`** does not work for applications with special requirements such as asynchronous event handling in our case. The correct approach to developing our RPC application is to generate prototype source code with the **`rpcgen`** tool and then modify the code to meet our special needs.

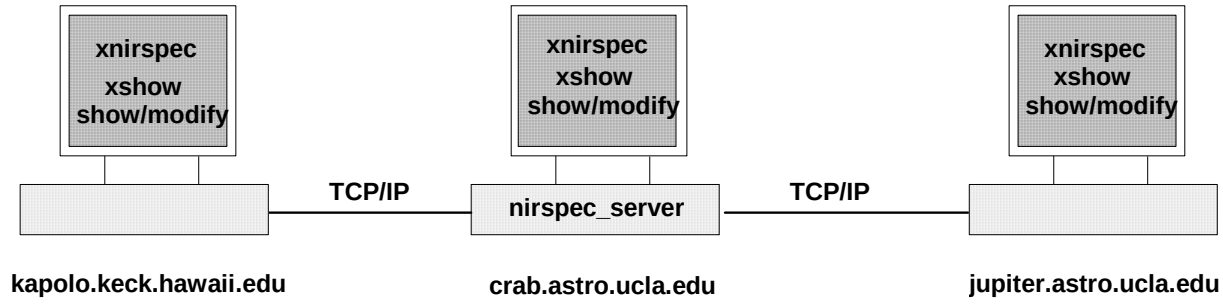

Figure 4 Client-server development and test platform

The NIRSPEC computer "crab" is used as both the server and the client platform. To provide a network platform to test the C/S code, we also use "jupiter", another SPARCStation in the Astronomy building, as a client site. It has IDL and a DataViews runtime license to provide the necessary environment. In addition, a "nirspec" account has been created on the Keck computer

“kapolo”. These three inter-networked computers provide us a complete development and test platform, as illustrated by Figure 4.