NIRSPEC

UCLA Astrophysics Program **U.C. Berkeley** **W.M.Keck Observatory**

Tim Liu **March 5, 1997**

NIRSPEC Software Design Note 12.00
Quick Look External Interface

1 Introduction

Like the other NIRSPEC user interface frond-ends such as the GUI and CLI, the quick look (QL) tool is also a client program under the NIRSPEC software client/server architecture. As Figure 1 shows, the QL communicates with the NIRSPEC server via the GUI. For instance, when the server broadcasts a frame ready keyword, the GUI will relay this message to the QL. There're two reasons why the QL interfaces to the GUI, rather than communicates with the server directly. First, sharing a single communication channel reduces the usage of network resources and therefore increases the efficiency in the client/server communication which is implemented using RPC (remote procedures call). Because the QL always runs together with the GUI, this scheme is apparently feasible. Second, unlike the GUI which controls the instrument and therefore must communicate with the server on a frequent basis, the QL doesn't need to receive and handle all the messages sent by the server. Therefore, with an indirect link between the QL and the server as shown in Figure 1, much of the incoming information can be filtered by the GUI and only those relevant messages such as frame ready signal should be re-transmitted to the QL. This will result in a more efficient QL program in its event handling.

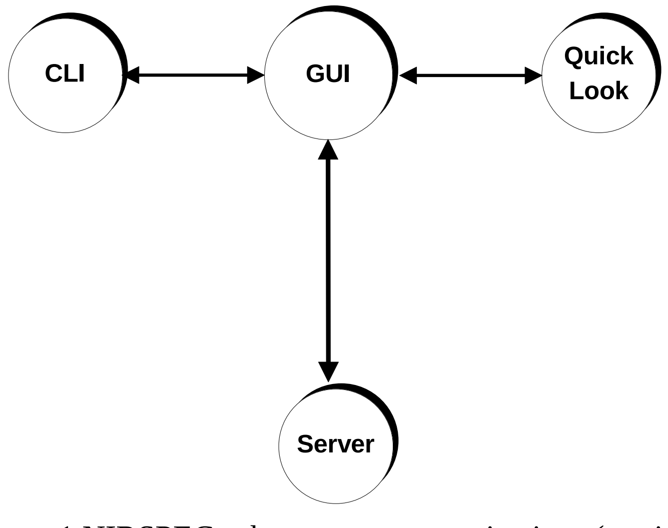

Figure 1 NIRSPEC sub-system communications (partial)

2 Requirements

The QL-GUI interface design must meet a few requirements. First, the communication scheme should be simple and easy to implement. As discussed above, the traffic handled by the QL is very light. In fact, the only time the server needs to talk to the QL is probably when a frame has arrived for display. Therefore, a complex communication system is obviously an over-kill. Although the interface should be simple, it must also be reliable. Second, the message flow between the QL and GUI should be efficient. It is noted that one end of the communication channel is handled by the GUI main loop and the GUI event handling is event-driven. Therefore, the GUI side of the communication is very efficient. However, the QL program uses a polling-based main loop to handle external events like communicating with outside. Thus, we need to have a fast polling frequency, while not compromise the performance of the QL program. This will be discussed in the next section. Third, the communication link should be bi-directional. While the GUI can send messages to the QL, the QL should also be able to initiate a talk with the GUI. For instance, the QL may need to request instrument configuration information from the GUI.

3 Structure

The QL is programmed in IDL widgets. There're several mechanisms for a IDL-based program to communicate with non-IDL software. In our case, the IDL CALL_EXTERNAL function is used to interface the QL to the GUI program. CALL_EXTERNAL allows an IDL program to call external functions written in other languages such as C from inside the IDL program and thus pass parameter values between two different software. To make this happen, an IDL event loop is constructed and the external C functions are called by the QL using a time interrupt. External function routines must be compiled as a shareable object library in order for an IDL program to call. In this way, a C-based programming interface layer is created for the IDL program.

Once the IDL program has a C external interface, the GUI which is programmed in C under X-window can communicate with it with a standard UNIX IPC (inter-process communication) mechanism. A socket is used to provide this function and such an interface scheme is illustrated in Figure 2.

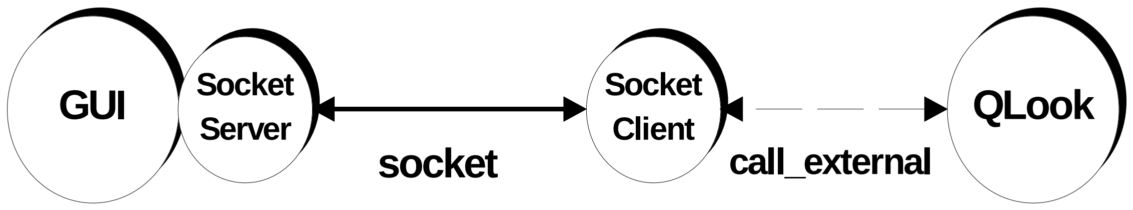

Figure 2 Interface scheme between the QL and the GUI

As the figure shows, the communication link consists of a socket server and a socket client. The socket server provides GUI-side interface routines, while the socket client is a shareable library containing socket I/O routines to be called by the IDL-coded QL. The communication scheme works as follows: when the GUI sends a message, either a keyword string from the NIRSPEC server or a special coded character string, to the QL, it calls an I/O routine in the socket server which writes the message string to the socket client. On the QL side, the event loop makes periodic CALL_EXTERNAL calls, and each time the socket I/O routine in the sharable library reads the socket channel and returns any message string if it is available. Similarly, If the QL wants to send a message to the GUI, the QL will pass the message as a function argument to the socket client with a CALL_EXTERNAL call. When the message arrives at the other end of the socket, the GUI asynchronous event loop will be notified and a socket server I/O routine will be invoked to read the message.

As mentioned, the QL event loop uses a time interrupt to poll the socket link for messages from the GUI. For an efficient handling of the message flow, the polling interval should be short. However, excessive function calls can downgrade the efficiency of the QL event loop and thus the performance of the QL program in carrying out other tasks. Tests show that the QL event loop is capable of making CALL_EXTERNAL calls at 10 Hz without any noticeable effect on the QL performance. Therefore, a 10 Hz or longer polling frequency should be ok.

When the NIRSPEC program is executed, the GUI will start the socket server and the QL starts the socket client with a CALL_EXTERNAL call. If the client is not up, the server will wait (20 seconds) for the QL to open the socket channel. Once the connection is established, it will remain open throughout the session.