SPEC Socket Programming

Logan Daum
Department of Physics, Massachusetts Institute of Technology, Cambridge, Massachusetts, 02139
(Dated: August 13, 2010)

In this project, a GUI for controlling motors and cameras was developed for the D1 station. The current setup at D1 used a stand alone video feed and a command line interface for controlling motors. The new GUI incorporates the motor controls into the existing video feed and adds new functionality to the GUI. The code was also given flexibility to be run with various motor configurations different than the two motor setup at D1.

I. INTRODUCTION

Currently at the D1 station, the method of interacting with the equipment inside the hutch involves multiple computer screens and mice while running several different programs. Preferably this setup could be optimized to run on a single computer while running fewer programs. The existing setup uses a MacCHESS crystal centering program for a video feed, a SPEC command prompt to interact with motors, and a third program to manage data analysis.

The goal of this project was to combine the video feed with the motor controls into a single program and facilitate data collection from the same window. The crystal centering program was to be modified to be capable of communicating with SPEC via sockets. The remote SPEC communication method would not only be done in Java, the language of the crystal centering program, but would also be implemented in C++ and Python for potential future use. The crystal centering GUI (graphical user interface) would also be upgraded to include new features. This new GUI would be able to be easily modified to be used at other beam lines with different motor configurations.

II. BACKGROUND

A. D1

The D1 hutch is primarily used for GISAXS (Grazing-Incidence Small-Angle X-ray Scattering) and GIWAXS (Grazing-Incidence Wide-Angle X-ray Scattering) on organic materials such as small aromatic molecules and conjugated polymers. The controls for D1 outside the hutch run two translational motors for the sample stage, the camera feed of the sample, and the CCD camera for diffraction data. Other sample stages with motors on different axes can also be used (Figure 1).

B. SPEC Server

SPEC is an X-ray diffraction and data acquisition software package created by Certified Scientific Software for UNIX based operating systems. It has built-in motor controllers which are capable of interacting with the motors at D1. SPEC is generally controlled with a

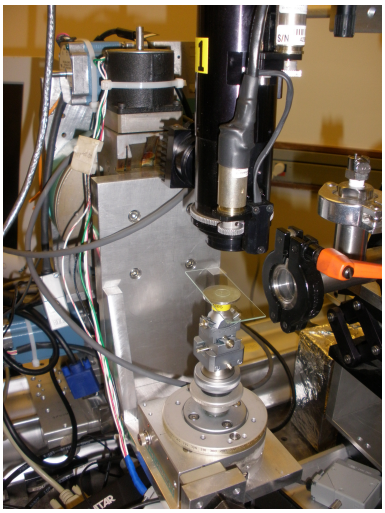

FIG. 1: Stage at D1 with two translational motors and a camera above

command line interface which uses syntax similar to C (Figure 2). It also supports macros and comes with an extensive macro library as well as custom made macros. Additionally SPEC can be operated in a server mode which is capable of interacting with other software remotely. This communication is done over network sockets by sending specifically formatted packets.

Communication with the SPEC server mode is done exclusively through TCP (Transmission Control Protocol) sockets. TCP sockets, as opposed to UDP (User Datagram Protocol), supports ordered error-free packets at the cost of higher latency. When SPEC is started in server mode it opens a TCP socket at a given port and listens for incoming connections on that port. A remote program can connect to SPEC by binding to the specific port SPEC is listening to. Once a connection is established, SPEC initiates three different threads: a thread listening and reading incoming packets, a thread sending packets, and thread executing SPEC commands. The SPEC command thread is the same thread which is used by the command prompt so both communication methods can be used concurrently.

Packets that are sent and received by SPEC must follow a specific format to be recognized. Each packet contains two main sections: the header and the data. The header contains information such as a send time, packet format type, command type, and property name, as well as the type, size, and format of the data section. The data section immediately follows the header and contains any extra information as described in the header. The command type in the header specifies the purpose of the packet, such as reading a variable, calling a function or macro, or replying to a query or event. Depending on the given command type packets do not always have a data section nor do they always specify a property name [1].

C. MacCHESS Crystal Centering Program

The MacCHESS crystal centering program used for the video feed at D1 was originally created for aligning crystals at F1 [by I. Degani]. This Java program was capable of controlling motors within the hutch in addition to camera controls and video feed. The motors could be move in three translational direction (X, Y, and Z) and a single rotational one (Phi). The GUI allowed for the motors to be moved via buttons and by clicking on the

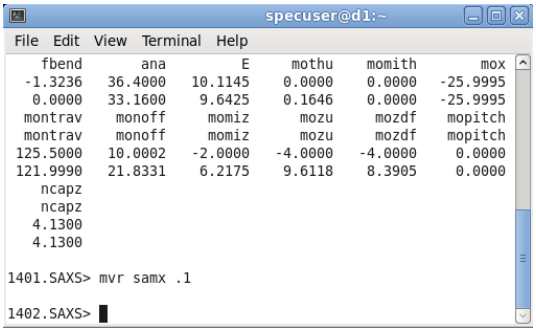

FIG. 2: SPEC command prompt

video feed to move to a specific location on the sample. The *Phi* motor can also be moved by a dial; the dial position is updated to the current *Phi* position whenever the *Phi* motor is moved.

The crystal centering program is based on a client-server model; the server interfaces with the motors and cameras while the clients are the GUI instances. The server initializes the motor controllers, camera feed, light controls, and restores previous session data. Once the server initialization has been completed, the GUI can be launched. The GUI, shown in Figure 3 features a central video feed with a motor control panel on the bottom left and camera controls on the bottom right.These controls communicate directly to the server when activated, which then directs the commands to the relevant hardware. The server is also responsible for updating each client GUI to any changes such as video settings and dial positions.

Within the GUI, each separate control group is integrated into the GUI as a plug-in. Each plug-in contains a set of source files which are inherited from a framework set of base classes. The framework base classes contain basic client-server methods such as sending messages to the server and updating the clients. Each plug-in has six main source files: Request, Update, Plugin, PluginFeatures, Panel, and PluginNotification. The Plugin file generally runs on the server side (some run client side) and is the main plug-in file which received commands. The Request and Update files define classes for passing messages between the client to the server and the server to the client, respectively. The PluginFeatures give the Plugin access other global variables, other plugins, and the main GUI components. The Panel runs client side and defines the behavior of the plug-ins GUI panel; it sends messages to server Plugin via Request objects. The PluginNotification receives all Updates and passes the message onto each client Panel.

Controlling motors via the crystal centering program is done through the CompuMotor plug-in. This plug-in contains the directional controls for the motors and is also capable of interfacing with the motors directly. Similar to the SPEC server, the crystal centering motors create a socket connections to communicate, which is how the CompuMotor plug-in interacts with the motors. In addition to the plug-in files noted above, it also contains a MessageHandler which receives packets sent from the motors and passes them onto either the server or clients.

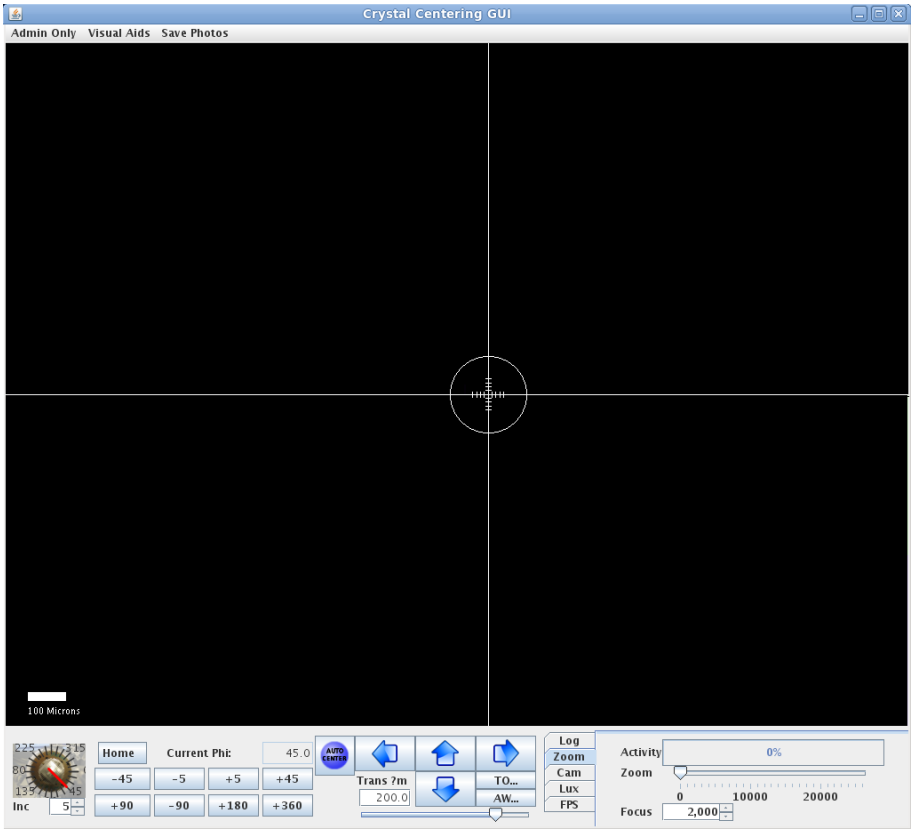

FIG. 3: MacCHESS crystal centering GUI

III. DEVELOPMENT

A. SPEC Server Communication

Interacting with SPEC servers was sampled in Python, C++, and Java. To test the server communication for each language, a socket connection with SPEC was first established and a binary data structure containing the header and data would be created and sent. The programs would then listen to the socket for SPEC replies, which could be generated from within the SPEC command prompt. Each language was successfully tested with the SPEC server, although the process was more involved with Java due to the lack of long integer types and struct class. These problems were solved by manipulating the bytes manually and by using a byte array.

B. SPEC Integration with Crystal Centering

In order to allow the crystal centering program to interact with motors at D1, the CompuMotor plug-in was to be replaced with a SPEC interfacing plug-in. This plug-in, SpecClient, would connect to SPEC when initialized on the server side of the crystal centering program. SpecClient also needed a MessageHandler similar to the CompuMotor MessageHandler in order to receive messages from SPEC. Since the communication with SPEC is done through binary packets, a new Packet class was created to hold both binary and Java

versions of commands. Constructors for the Packet class take either binary packets or Java commands and generate a Packet object that contains both types. This allows for easy conversion between the binary data and the Java commands.

An example of sending a request to SPEC would be whenever a move button would be pressed on a SpecClient Panel (Figure 4). In this case the Panel would generate a Request object which would be sent to the Plugin. The Plugin would parse the Request object and create a Packet object which could then be sent to SPEC. An example of the reverse process would be if the *Phi* motor position was changed which would update the GUI *Phi* dial. In this case whenever the motor position changed, SPEC would send a packet which would be received by the MessageHandler. The MessageHandler would create a Packet object from the binary packet and read the Java command strings. The Java commands would then be parsed and an Update would be sent to the PluginNotification. The PluginNotification would then call update functions within the Panel to move the dial.

FIG. 4: SpecClient Panel

C. New GUI Features

Final GUI is shown in Figure 5. The SpecClient Panel is shown in the bottom left while the ZoomFocus panel is in the bottom right. In place of the "autocentering button in the SpecClient Panel, which is not used at D1, two buttons have been added for controlling the shutter and taking CCD images. This GUI was successfully tested through interacting with the SPEC server at D1 while controlling the two translational motors.

In addition to changing the CompuMotor plug-in to the SpecClient plug-in, a few new features to the GUI were added. Since the motors at D1 could change while using different stages, the ability to change the motors assigned to each axis within the GUI was incorporated. This option was added to the Admin menu at the top of the main GUI frame. When this option is selected, a dialog box is opened which gives options for X, Y, Z, and *Phi* motors (Figure 6). The SPEC server is then polled to get the name of every available motor. Each axis can then either be assigned a motor name with a scaling factor or can be disabled. Once the motors have been selected, the motor names and scaling factors are stored in the server side Plugin class. These variables are also backed up in a database file which restores the variable values when the program is reopened. Since the *Phi* dial must also be updated when the motor is moved, an extra call is made to the SPEC server to monitor changes to the *Phi* position variable. Currently at D1 only the X and Y axes are being used for moving the stage so SPEC would not have to monitor any variables.

In addition to adding the ability to choose motors, user defined preset values for the camera zoom and focus were added to the ZoomFocus plug-in. Originally this plug-in allowed the zoom and focus to be set by a slider and a spinner box respectively. However since the zoom and focus are often coupled, the plug-in Panel was changed to have three preset zoom/focus settings available to be saved (bottom right of Figure 5). This can be done by assigning the desired preset values for zoom/focus and then pressing the "save

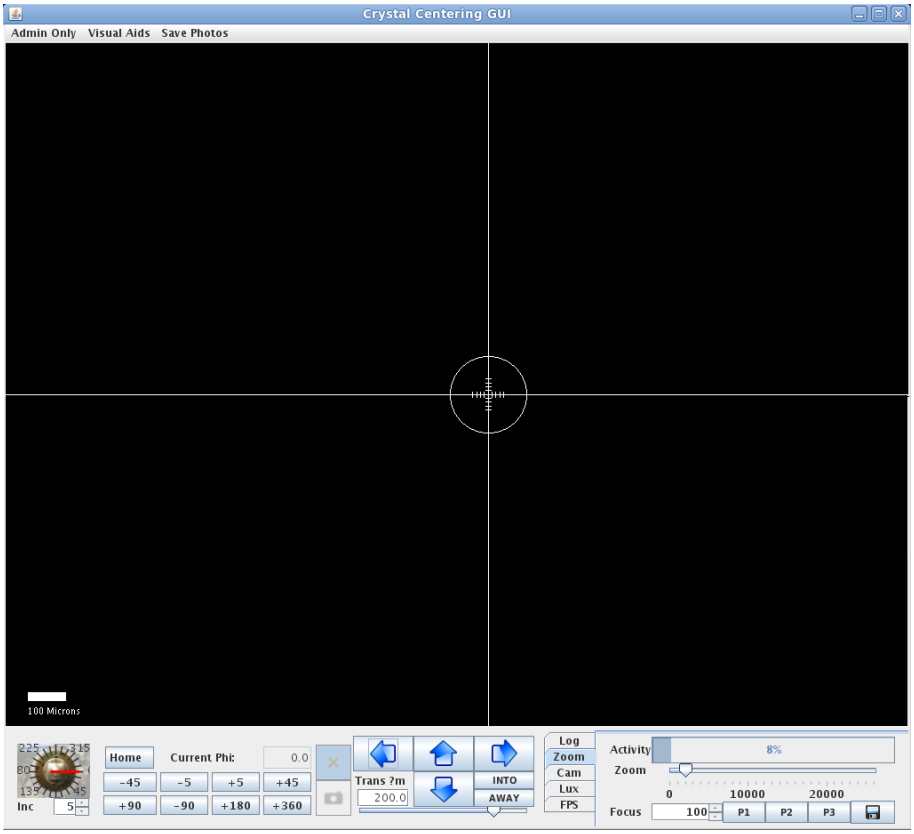

FIG. 5: Modified crystal centering GUI

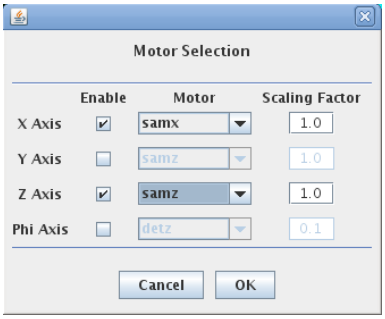

FIG. 6: Motor selection dialog box

button followed by the chosen preset button (“P1, “P2, or “P3). The stored values can then be recalled by pressing the preset button. The current preset values are shown in the tooltip.

IV. CONCLUSION

In summary, the sample SPEC socket programs were all able to communicate with the SPEC server. The MacCHESS crystal centering program was modified to be able to interact with the SPEC server to control motors and tested at D1. These motor controls were generalized for use at beam lines other than D1. Furthermore the ability to choose the motors for each axis and the zoom/focus preset values were both added to the GUI.

V. FUTURE WORK

Due to time constraints on this project, a two of GUI features did not get fully implements into the GUI. One of the features was the ability to manually control the shutter on the beam line via a toggle button. This feature was partially implemented with the "shutter button included in the SpecClient Panel, shown as an "X to indicate the shutter is closed (Figure 4). However the SPEC macro to open the shutter was not included in the code since at D1 the shutter is automatically controlled and the button was disabled. In order to add the shutter control, the button would need to be enabled and the macro command would have to be entered. When the button is toggled on the "X is change to a "check mark.

Another feature that was not fully implemented was being able to take pictures with the CCD camera and the stage camera at the same time. This feature could be selected by pressing the "camera button (disabled currently) in the SpecClient Panel, which is directly below the shutter button. When this button is pressed, a dialog window would appear with options to choose the exposure time on the CCD camera and a filename to save to. When the options are set and the "capture button is pressed, commands would be sent to both cameras to take pictures, which would then be save to similar filenames.

VI. ACKNOWLEDGEMENTS

I would like to thank Dr. David Schuller, my advisor, for guiding me through this project. I would also like to acknowledge Dr. Detlef Smilgies, Dr. Darren Dale, and Phil Sorenson for their help. Lastly I would like to thank the CHESS staff and the CLASSE REU program for this opportunity. Funding for this project was from the National Science Foundation REU grant PHY-0849885 and DMR-225180.

---

[1] *SPEC*. Certified Scientific Software (2008). Available: `http://certif.com/spec.html`