Controlling Scientific Instruments with JAVA

University Of Rochester Laboratory for Laser Energetics
Advised by Dr. Christian Stoeckl
David Bowen

Abstract

Recording electrical waveforms on OMEGA is a task that has usually required a computer for controlling oscilloscopes that capture data from the diagnostic equipment. With a new generation of oscilloscopes integrated with a fully functional computer, the oscilloscope can run control software itself, removing the need for a control computer. However, the control software, written in JAVA, must be changed and updated to be operational on this new generation of oscilloscopes. By changing the interface layers of the software, it is possible to apply these changes to all diagnostics software instead of being particular to one program. This means that all of various control programs could easily be updated to operate on the newer oscilloscopes. The software was changed to have more reliable treatment of interface errors and was also changed graphically as to appear friendlier to new users.

Introduction

The concept of laser fusion, like the type attempted here at LLE, uses the inertia of an exploding shell to compress two isotopes of hydrogen, Deuterium and Tritium, to high density and high temperature. Some of the Deuterium and Tritium atoms combine to form an intermediate particle immediately breaking down into an Alpha particle and a neutron. As the unstable particle breaks down it releases energy. An important part of the diagnostics of laser fusion is measuring the amount and energy of neutrons escaping the chamber. Though neutron radiation is not the only diagnostic being performed on OMEGA, Dr. Stoeckl was a neutronics scientist so neutron detection was the diagnostic I was made familiar with. These neutrons are measured using scintillating detectors. A scintillating detector consists of scintillating plastic connected to a photomultiplier tube. Neutrons emitted from the chamber hit the scintillating plastic and strike protons in the hydrocarbon molecule of the plastic. The elastically scattered

protons, the neutrons excite the molecules causing them to emit photons, or scintillate. The photons strike an extremely negatively charged photocathode in the photomultiplier tube, causing a serial amplification by knocking one electron into the next plate, knocking more electrons into the next plate, and so on until the photomultiplier tube delivers the resulting charge to an oscilloscope. The signal can then be recorded by an oscilloscope and transferred into a computer for further analysis.

The computer system for controlling oscilloscopes is a bulky and demanding setup. A computer must be connected to the scope using an interface called a GPIB or General Purpose Interface Bus. The only way for a computer to use this connection successfully is if the computer is fitted with a GPIB card. So a computer must not only be loaded with all of the correctly debugged software necessary for control, but also an interface card and its driver software. The computer would also require network capabilities to receive pre-shot notifications. So to receive data properly, the above system would have to be added to the data acquisition location. Though this may not seem to be the worst of possible data acquisition situations, it can be improved.

The most recent models of oscilloscopes have been designed to be a computer operating in parallel with an oscilloscope. The computer side of the newer models runs an operating system that, by default, runs the software that displays the oscilloscopes data. This new generation of oscilloscope is capable of replacing the previous two-machine setup. The oscilloscope would have no need for GPIB hardware and the scope is already fully equipped with networking capabilities. The computer could run the control software in parallel with the oscilloscope and remove the need for any added hardware.

Though conceptually the idea of running the control software on these new oscilloscopes seems a reasonable idea, it requires some interesting reverse engineering. The problem lies with changing all of the control software, which was written to interface with GPIB cards and GPIB cables to an external oscilloscope, so that it now interfaces with the oscilloscope which functions as a part of the new generation of computer/oscilloscope hybrids. Before the potential of these new computer/oscilloscope hybrids can be fully taken advantage of, the current software has to be updated so it can smoothly go from an internal environment to an external GPIB interface.

Learning the structure of the Software

The existing software had been written for GPIB connections. However, now the

software must run on a computer talking to the scope not with GPIB, but through some sort of internal interface.

So, where would this new interface start? Where was the backdoor that could be used to access the data the oscilloscope was capturing? It turns out that the answer lied within software provided by Tektronix® .

The software engineers at Tektronix had written a software interface to allow their own analysis software to run on the scope and to access its data. All one would have to do, is to use the methods of the classes defined by Tektronix® to talk to the internal scope.

From there, I proceeded with further examination of the code of the program and the supplied classes. The program I was using specifically was ScopeControl, a good choice as it was designed for single scope acquisitions. The situation was even better, though, because the software had been written in layers. Each layer was a link from one part of the program to a lower part. The part that I had to update, the interface, was at the very bottom. The layer, or class, named Gpib contained the methods for communicating with the oscilloscopes through a GPIB connection. Since all control software uses the same layering, changes could be made in one copy of Gpib.java, tested using ScopeControl, and then distributed amongst all control programs. However, Gpib now had to be two different classes; one to accommodate internal interface and one to accommodate GPIB interfaces for a program to truly be useful and work with both interfaces.

To first write an interface based on this software by Tektronix®, we must first know what this software contains. Unfortunately, Tektronix® did not supply us with source code. So, to view the contents of the software, it was necessary to disassemble the classes. From doing this, I obtained the knowledge that the class GpibDevice appeared to be the super class of InstGpibBus. GpibDevice seemed to contain mostly private methods and its constructor required an object of InstGpibBus. InstGpibBus revealed the public methods required for the program to operate and communicate properly with the scope. My task was to synthesize a Gpib class with method definitions redefined for the classes provided by Tektronix®. So, by analyzing the disassembled code, I learned about methods that performed the same operations that were required by the interface; *read, write, clear, get,* and *status.*

With the redefined Gpib class and some routine debugging, ScopeControl was able to communicate internally with the scope, using Tektronix® supplied access classes and .DLL's.

Versatility

At this point, I had successfully created an internal interface, though not efficient or reliable. It was not quite of the quality where it could be reliably used and distributed. In fact, it was certain that the current version of the program would simply not work with a GPIB connection. Further extension was necessary.

A quality that seems to be the mark of a well-written program is automation. The less that the user must define, the more user friendly the program is, the better the program is. However, research may continue into the automation of the GPIB connection selection. As of now, the GPIB connection type is a user defined setting and will result in error if selected incorrectly.

The design of the Gpib setup is another interesting part of this versatile program. There exists the new internal version, *Gpibint* (see figure 1 for a diagram of the structure of the new internal interface), and the external GPIB cable version, *Gpibext*. They were tied together as extensions of a super class, *Gpib*. *Gpib* contained the methods common to both *Gpibint* and *Gpibext*. However, the program still must be told which class file to use. On the user side, this is done with a check box in the settings menu.

ScopeControl

The program that I subjecting to the testing of the internal interface was a program called ScopeControl, its purpose being for a computer to capture data from an oscilloscope, either once or repeatedly as with a log or a run. However, with the new type of oscilloscope capable of writing its data to a hard drive or network drive, many of the functions of ScopeControl are not necessary. The only remaining useful function would be its ability to take and record data in intervals, known as logging. However, the very design of ScopeControl made it an ideal program with which to do testing with scope interfacing. The functions were clear and user executed making this an instant results type of program.

By using ScopeControl and changing it's layers, the goal was achieved of creating the internal interface and making it reliable. Updating ScopeControl was more a tool of education of software layering and design rather than a practical expenditure (see Figure 2 and Figure 3 for screenshots of ScopeControl's user interface before and after the updates). The redesign of ScopeControl did, however, create a user friendly and reliable example of the internal interface.

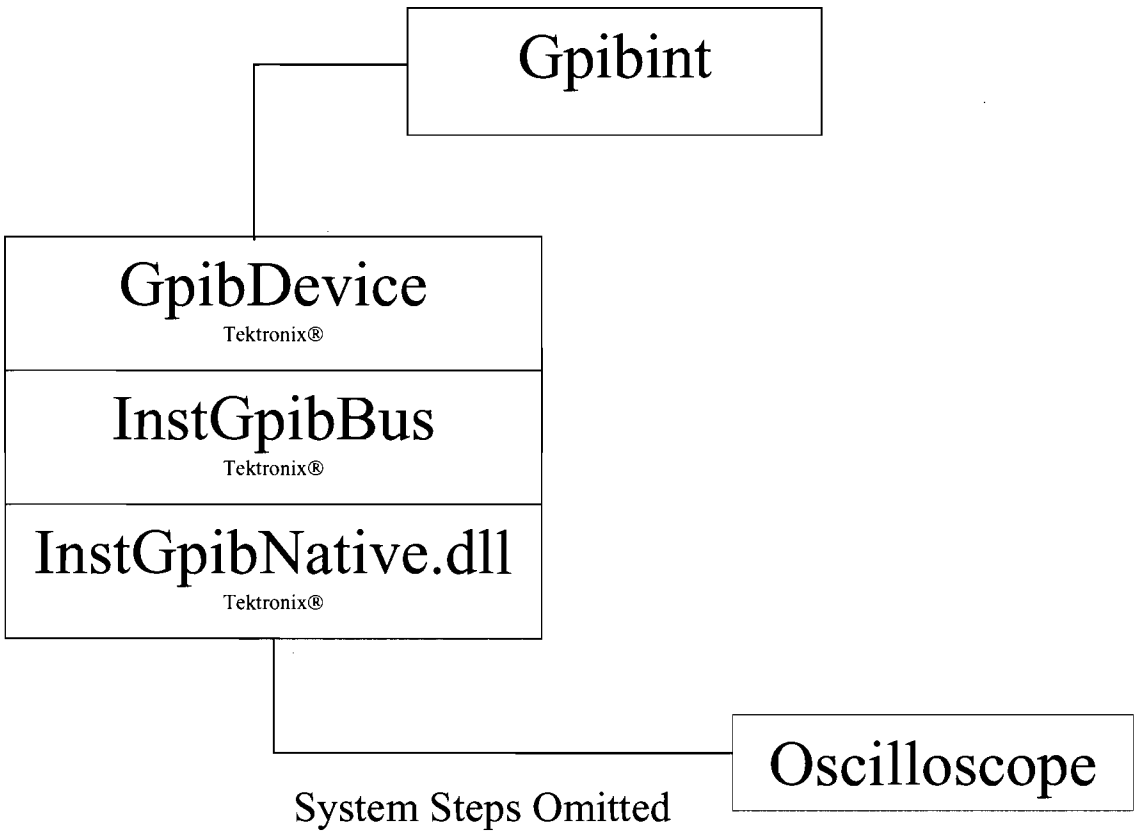
Conclusion

By changing the layering structure of diagnostic software, such as ScopeControl, a single version of any diagnostic program can be made to run on either a control computer or on the newer computer-oscilloscope hybrids. ScopeControl was updated to exemplify the ability to make diagnostic software not only versatile but also reliable and user friendly. ScopeControl was only an example. The low level interface software developed in this project will be used by many applications running on the new generation of oscilloscopes. This will end the necessity of a separate computer to archive data collected by previous generations of oscilloscopes on OMEGA.

Figure 1

ScopeControl internal interface structure

As the diagram shows, the Gpibint layer uses methods defined in InstGpibBus.class to interface with InstGpibNative.dll which in turn communicates with the oscilloscope.



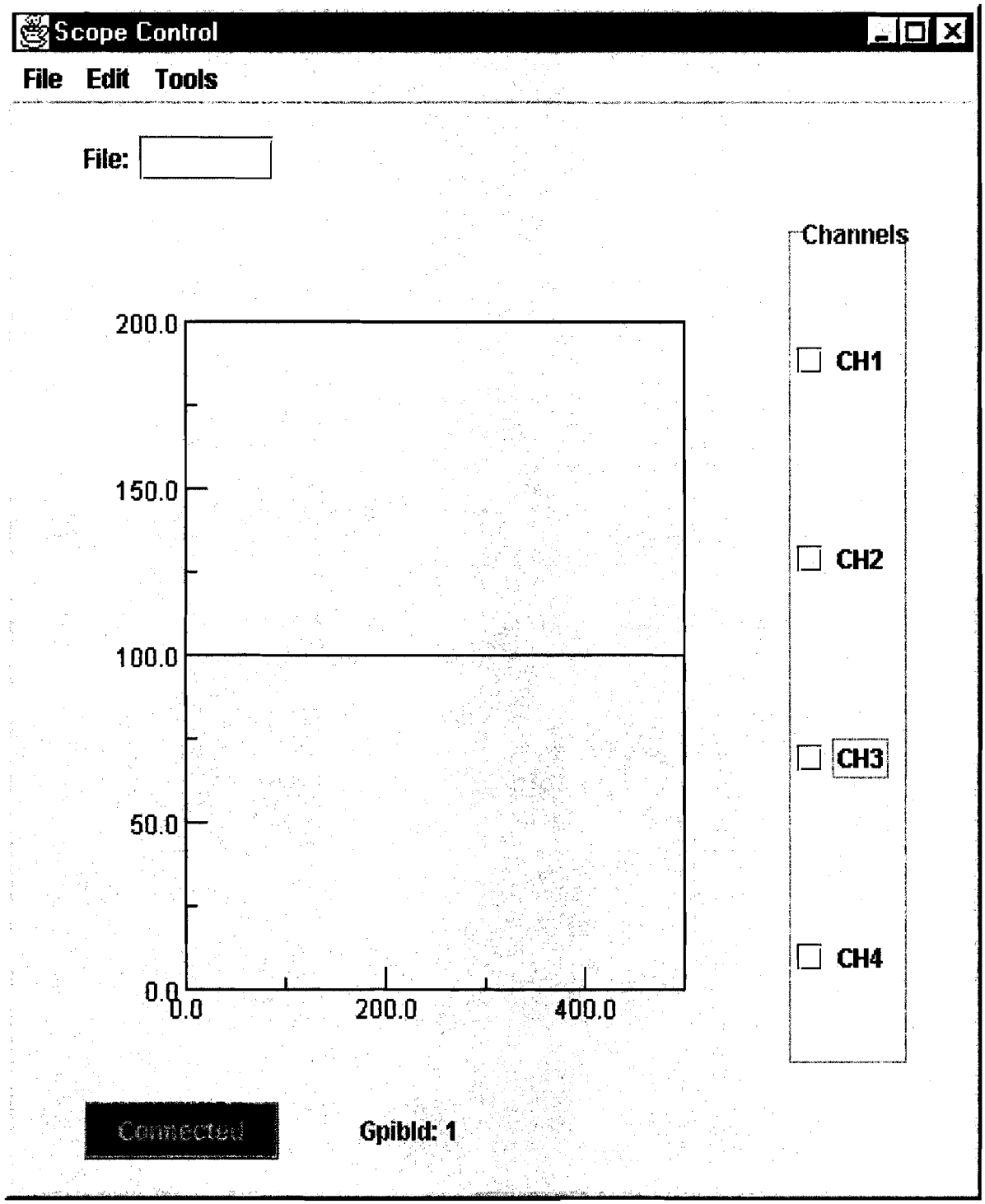


Figure 2.

ScopeControl

July 9th, 2001

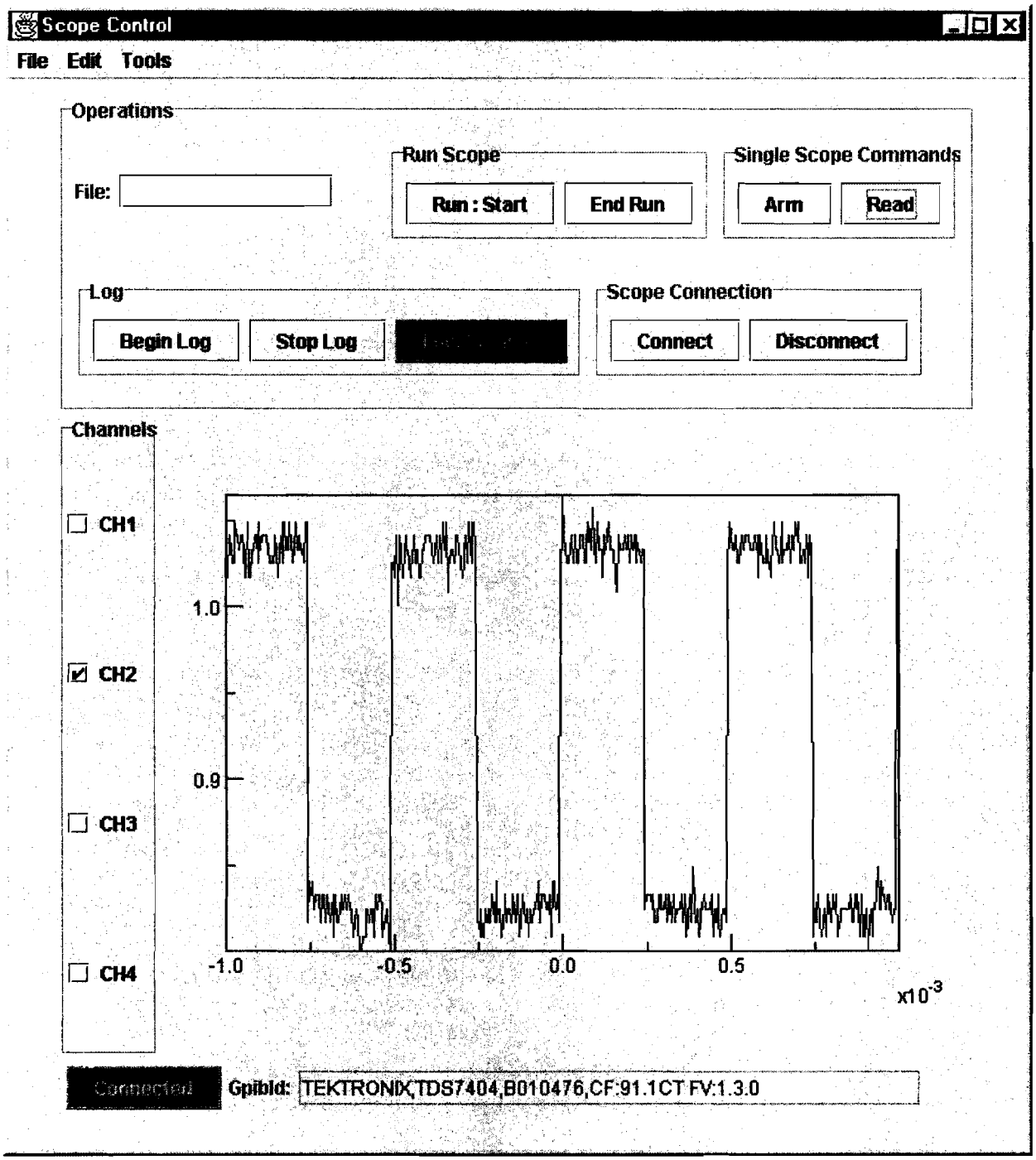


Figure 3.

ScopeControl
Modified by Dave Bowen
as of August 31st , 2001

Appears with updated user interface and underlying error messaging improvements.