Triana - A Quicklook Data Analysis System For Gravitational Wave Detectors

Ian Taylor[1],Bernard Schutz[2]

[1]*Department of Physics and Astronomy, University of Wales Cardiff, PO BOX 913, Cardiff, Wales, UK*
[2]*Albert Einstein Institute, Max Planck Institute for Gravitational Physics, Schlaatzweg 1, Potsdam, Germany*

Abstract. In this paper, we present an overview of the Triana OCL (object connecting language) data analysis environment. Triana is an extension of Grid, introduced last year, that allows users to interactively build complex data processing systems by selecting a set of desired tools and connecting them together graphically.

1 Introduction

Signal-processing systems are becoming an essential tool within the scientific community. This is primarily due to the need for constructing large complex algorithms which would take many hours of work to code using conventional programming languages. Triana (Object Connecting Language) is a graphical interactive multi-threaded environment allowing users to construct complex algorithms by creating an object-oriented block diagram of the analysis required.

2 An Overview

When Triana is run three windows are displayed. A *ToolBox* window, a *MainTriana* window and a *Dustbin window* (to discard unwanted units). Figure 1 shows the *ToolBox* window which is divided into two sections. The top section shows the available toolboxes (found by scanning the toolbox paths specified in the Setup menu) and the bottom shows the selected toolbox's contents. Toolboxes (and associated tools) can be stored on a local server or distributed throughout several network servers. Simply adding the local or *http* address in the toolbox and tool path setup allows on-the-fly access to other people's tools.

Units are created by *dragging* a unit's icon from the ToolBox window and *dropping* it at the desired position in the MainTriana window. Units are then connected together by dragging from an output socket on a sending unit to an input socket of the receiving unit. Algorithms are run by clicking on the *start* button (see Figure 2) located at the bottom right-hand side of this window. Algorithms can be run in one of three modes : in a *single step* fashion (i.e. one step at a time); *continuously* (i.e. each unit running continuously) or in

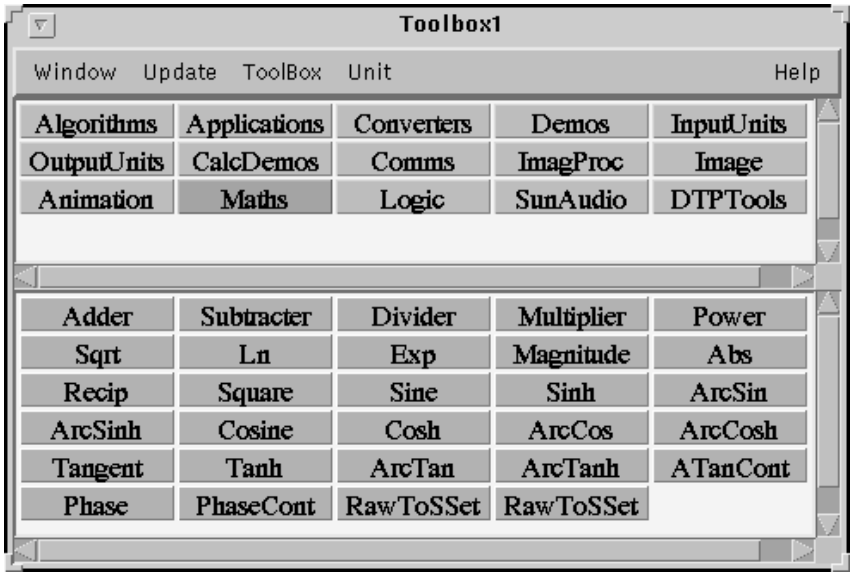

Figure 1: Triana's ToolBox window. Toolboxes can be organised in a similar way to files in a standard file manager.

continuous step mode where the units are run continuously but synchronised at each time step.

2.1 Overview of the Toolboxes

The following toolboxes are currently available within Triana :-

- *Algorithms :* this contains a number of signal processing routines e.g. FFTs, amplitude and power spectral algorithms, averagering and statstical units, noise adders (i.e.Gaussian), complex number routines, automatic number padders and frequency low- and high-pass filters.
- *Applications :* this currently contains MathCalc (see section 4) but it will later be used to incorporate other applications into Triana.
- *Converters :* converting units allow the user to convert between the various types within the Triana system. For example, a user could load in a set of raw samples using the Importer unit (i.e. into RawData object) and then pass this RawData into a RawToSSet unit in order to specify the sampling rate and add a description etc.
- *Demos :* various demos of connected networks (i.e. groups) within Triana. Used to demonstrate to users how the various units are connected together.

We also have another demo toolbox, called **CalcDemos**. This contains examples of using and connecting the MathCalc unit within networks.

- *InputUnits :* these are units which generate data somehow or import data into Triana from other sources e.g. files. We have wave generators, text and binary importers, 2-dimensional data set importers and generators and a unit to generate a constant (useful for setting parameters for other units).

- *OutputUnits :* these are units which output data somehow or display data. Here we have text and binary exporters, 2-dimensional graphical displayers, a Grapher (for displaying sample set's and spectra) and viewers for constants (a simple text field view) and text (i.e. via an editor we've created called *Ved*).

- *Comms :* consist of units (*Client and Server*) which allow two versions of Triana to communicate with each other via internet sockets and a unit (*Exec*) which can run and communicate with any executable . Exec communicates with executables by using standard input and output streams.

- *ImagProc :* contains various image processing algorithms. There exist units which can alter the brightness, colour intensities, convert from colour to black and white, edge detection, contrast enhancers, blurrers, convolvers, image adders and subtracters, image negaters etc.

- *Image :* consist of a set of tools which allow the loading in and displaying of GIF or JPEG files.

- *Animation :* here we have a set of tools which allow you to animate a set of GIF or JPEG files and control the animation speed.

- *Maths :* consists of a set of standard mathematical functions which work on the various types within Triana e.g. Ln, Sqrt, Exp, Magnitude, Square, Sine, Sinh, ArcCos etc.

- *Logic :* currently just contains an If unit which tests its first input and routes the second input to either the first or the second output according to the outcome of the test. Other logic units are planned for the final release of Triana.

- *SunAudio :* contains a demo unit which allows users to speak or sing (!) into their Sun workstation have this input into Triana.

- *DTPTools :* these are a set of tools which allow a number of desk-top publishing tasks to be performed within Triana. There is a general way of producing listings of files (using wildcards) throughout subdirectories. This means that you can process *all matching files* with any of the tools

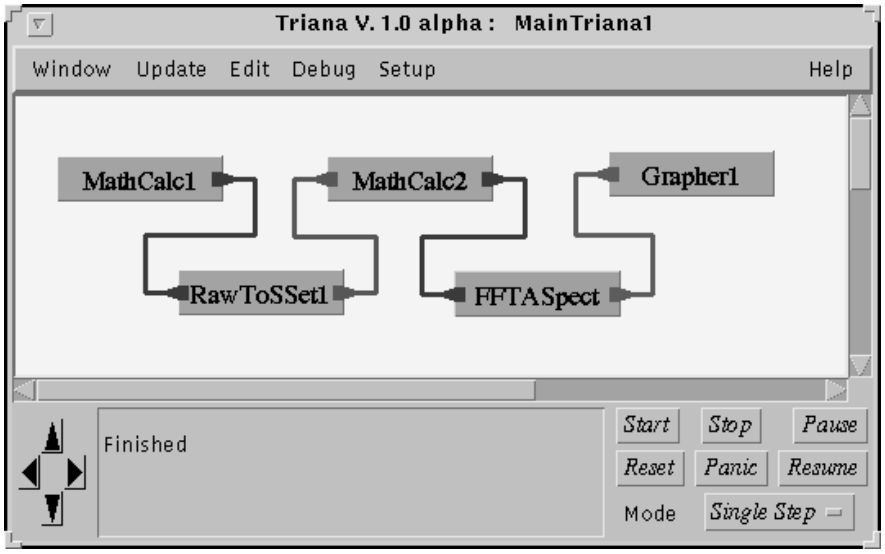

Figure 2: A snapshot of Triana's programming window.

within this toolbox. For example, you could count the lines of all the .java code within a source directory and all its subdirectories, change all occurances of one word with another or even list files (with line numbers) of all files containing a phrase (like grep).

3 New Features Of Triana

Groups of units can now be saved along with their respective parameters. Such groups can also contain groups, which can contain other groups and so on. This is a very powerful feature which allows the programmer to hide the complexity of programs and use groups as if they were simply units themselves. Many improvements have been made to the graphical interface, including compacting the look and style of the toolbox, adding a snap-to cable layout and many more informative windows. The major change however, is that now Triana consists of an object connecting language (OCL) and a separate user interface. This means that the units can be run from within the user interface or as a stand-alone program.

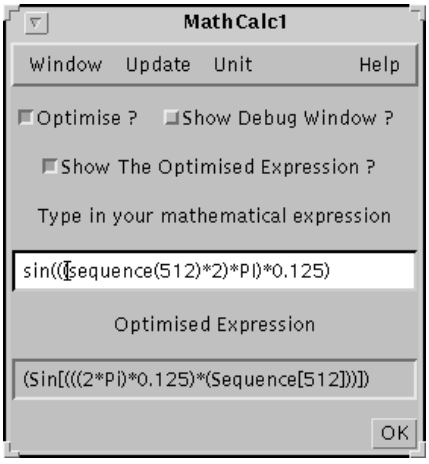

Figure 3: MathCalc1's parameter window

4 MathCalc

The MathCalc unit interprets, optimises and evaluates arithmetic expressions using stream-oriented arithmetic. It recognises a large number of functions and constants. It can be used to evaluate scalar expressions, to process input data sets, or to generate output data sets. All calculations are performed in double-precision real arithmetic.

Stream-oriented arithmetic can be defined as the application of an arithmetic expression to each element of a stream independently. Thus, if B is the sequence $b1, b2, .., bn$, then the function $sin(B)$ evaluates to the sequence $sin(b1), sin(b2), ..., sin(bn)$. MathCalc distinguishes between *constants* and *sequences* or sets. Sets (data sets) are sequences of numbers and constants are single numbers, essentially sequences of length 1. In a MathCalc expression the two can be mixed very freely, with the restriction that all sequences must have the same length. Sequences or constants can be obtained from the input nodes of the MathCalc unit. The example given in the MainTriana window (see Figure 2) demonstrates the flexibility of the MathCalc unit.

The first *MathCalc* unit creates a 125 Hz sine wave by using the equation $sin(((sequence(512) * 2) * PI) * 0.125)$ where the sample rate is 1kHz (MathCalc will optimise this to sin[(2*PI*0.125) * sequence(512)], see Figure 3). MathCalc's user interface allows users to type in any mathematical expression. This expression is then optimised (when the user hits return) and the optimised expression then appears in the bottom text field. The user can choose to show the optimised expression or to hide it (see Figure 5 for the MathCalc parameter window without the optimised expression shown.)

This sine wave is then transformed into a SampleSet type by adding its

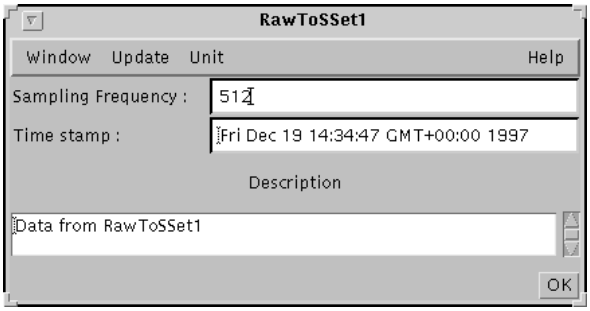

Figure 4: RawToSSet1's parameter window

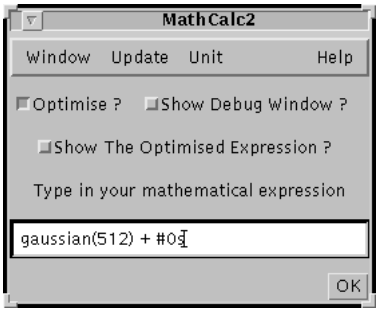

Figure 5: MathCalc2's parameter window

sampling frequency (i.e. 1 kHz) within the RawToSSet unit (RawToSSet1, see its parameter window within Figure 4).

The second MathCalc unit adds Gaussian noise to its input (i.e. by typing $gaussian(512)+\#0s$ in MathCalc2's parameter window, see Figure 5) The #0 means node 0 and the s means that it is a sequence as opposed to a c which would a constant.

This noisy signal is then passed to the FFTASpect group (see Figure 6). This group consists of an FFT unit (FFT1) and a unit which converts the complex set of data, generated by the FFT, to an amplitude spectrum (AmpSpect1)

Groups, such as the FFTASpect can be created very easily within Triana. Simply by choosing the units to group and selecting *Group* from the *Edit* menu or from a pop-up menu will group the units and create a Group edit window for them. This can be performed again within the group edit windows to have groups of units within groups of units and so on. Very complex algorithms can be created and broken down in this way. This amplitude spectrum is then displayed by Grapher1 (see Figure 7).

Once the signal is displayed, it can be investigated further by using one of the Grapher's various zooming facilities or displayed differently by choosing one

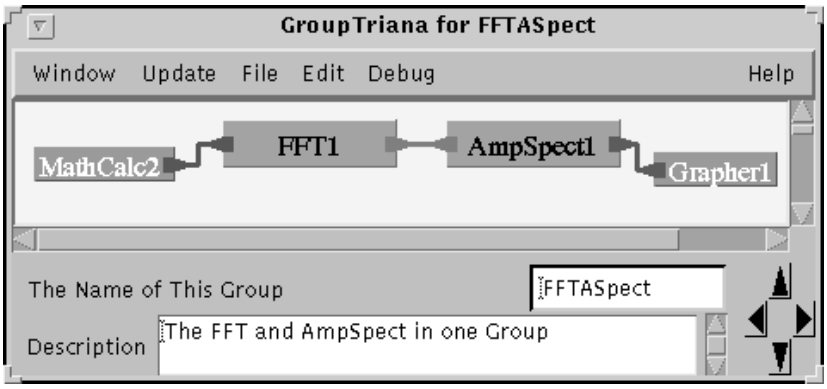

Figure 6: The internal structure of FFTASpect's group. The outside units (mathCalc2 and Grapher1) are simply labels indicating which units they are connected to in the next level up i.e. within the MainTriana window (see Figure 2)

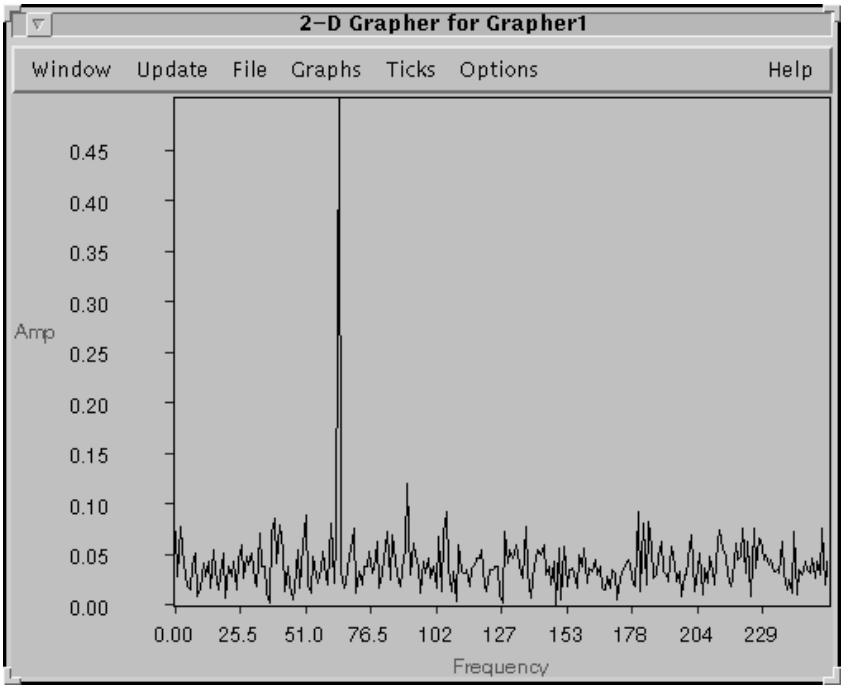

Figure 7: Grapher1's output

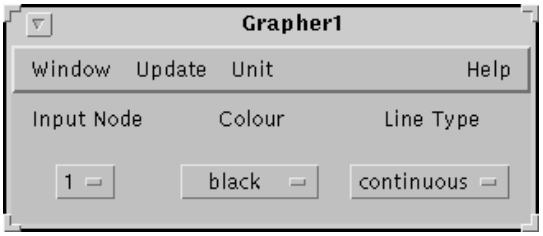

Figure 8: Grapher's parameter window

combination of line styles or colours. Any grapher can simultaneously display many signals (up to 20 nominally) each with its own colour and line style. This can be set using the Grapher's parameter window shown in Figure 8 below. Each input node to the grapher can be displayed in one of thirteen different colours and in one of three different line styles : continuous (as in Figure 7), scattered (i.e. just dots for each point) or impulse (vertical lines from the point to the x-axis).

Zooming can be controlled via a zoom window which allows specific ranges to be set (see Figure 9) or by simply using the mouse to *drag* throughout the image. For example, by holding the control key down and dragging down the image is zoomed in vertically and by dragging across from left to right zoomed in horizontally. The reverse operations allow zooming out. Also once zoomed in, by holding the shift key and the control key down the mouse can be used to move around the particular area you are interested in. We also have another powerful zooming function which literally allows the user to drag to the position of interest and the image will zoom in accordingly.

Using the zoom window, precise zoom regions can be specified by inserting the desired ranges in the top four (north, east, south and west) textfield boxes. There are options to retain the current zoom ranges when displaying further images and to specify which directions you want to zoom into. These take effect when the mouse-dragging functions are used. The two buttons at the bottom allow the user to zoom into 50% of the image (*Half Size*) or to return to the full size of the image (*Full Size*). Other useful features of the Grapher are to be able to change the distance between ticks and to view the data logarithmically in either vertical or horizontal directions.

5 Current Version Of Triana

Triana originated from an implementation of the system using C++ and InterViews [1] but was abandoned in early 1996. Version two [2] was written using the Java Development Kit, JDK 1.0.2 but this was updated in order to be compatible with the new JDK 1.1.x kit. We also re-implemented the base classes

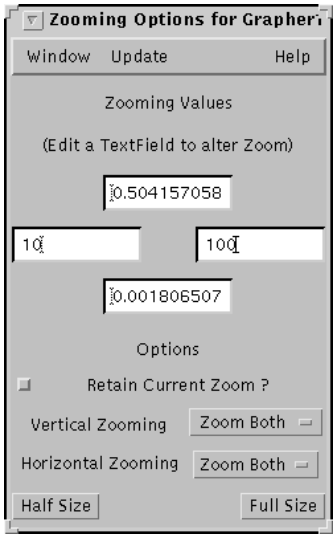

Figure 9: Grapher's zoom window.

and created OCL. This version of the software was called Grid OCL [3] but in November 1997 we changed its name to Triana which is now a UK trademark. It is a late *alpha* version and all information relating to the software can be found on the internet site, *http://www.astro.cf.ac.uk/Triana/*. Triana OCL will be entering its beta testing stage early 1998 followed by a final version shortly after. We are in the process of being able to provide a commercial version of the software for which support can be given. None-the-less we will always provide it in a free downloadable from the WWW with a certain time limit (3 or 4 months) and to the gravitational-wave community. Our main goal is to create a very wide user base and get people involved in writing and using there own units as well as our own.

So far, collaborators are working on tools for various signal and image problems, multimedia teaching aids and even to construct a musical composition system.

References

[1] Taylor I. J., & Schutz B. F., 1995. The Grid Musical-Signal Processing System, International Computer Music Conference, p. 371-371.

[2] Taylor I. J. & Schutz B. F., 1996. The Grid Signal Processing System. Astronomical Data analysis Software and Systems VI, p. 18-21.

[3] Taylor I. J. & Schutz B. F., 1997. Grid OCL : A Graphical Object Connecting Language. Astronomical Data analysis Software and Systems VI, p. 18-21.