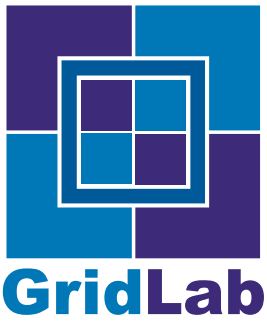

IST-2001-32133

GridLab - A Grid Application Toolkit and Testbed

D3.4 Triana Release

Author(s):	Ian Wang, Matthew Shields and Ian Taylor
Document Filename:	GridLab-3-D3.4-0002-Release
Work package:	WP3
Partner(s):	Cardiff University
Lead Partner:	Cardiff University
Config ID:	GridLab-3-D3.4-0002-1.0
Document classification:	Public

Abstract: This document outlines the open source release of the Triana. Within this document we outline how the Triana release can be downloaded over the Internet, and also the salient features of the release with regards to the GridLab project.

Contents

1 Introduction

At the start of June 2003 a new open-source version of the Triana workflow-based problem solving environment was released on the Internet. This release constitutes a large rewrite of the previous version of Triana, with a significantly improved user interface and task model. The new version also allows the user to distribute their workflow using Grid technologies, and to incorporate existing Grid services into their workflows. It should be noted that this release is a beta-version, and thus is not expected to be bug free. We hope that feedback from users of this release will help us improve Triana.

The release itself fulfilled the deliverable; this document is only intended as a guide to downloading and running Triana, and as an overview of some of the new features. In Section 2 we describe how to obtain the Triana release, before giving an overview of Triana and its distribution mechanisms in Sections 3 and 4. In Section 5 we introduce the GAP Interface, our stop-gap middleware independent grid interface that we are using while we await a Java version of the GridLab GAT [Allen et al., 2002]. We summarize this Triana release in Section 6.

2 Downloading Triana

The source for Triana can be downloaded from:

```
http://www.trianacode.org/projects/triana/
```

Available from this site is the latest stable build, and the latest nightly drop. Read and write CVS access to Triana is also available by emailing robert.davies@astro.cf.ac.uk.

Also required to run Triana is Java 1.4, which can be downloaded from:

```
http://java.sun.com/j2se/downloads.html
```

As Triana is written in Java it should execute on any machine with a Java virtual machine. Before you build Triana ensure that the environment variable TRIANA is set to the triana root directory, and that JAVA_HOME is set to the java root directory. To build triana on any platform go to the Triana bin directory and type:

```
buildTriana
```

To launch Triana type:

```
triana
```

To launch a Triana launcher service (see Section 4.1) as a P2PS or JXTA peer type either:

```
trianaservice -p2ps
or
trianaservice -jxta
```

More complete instructions can be found on the trianacode website.

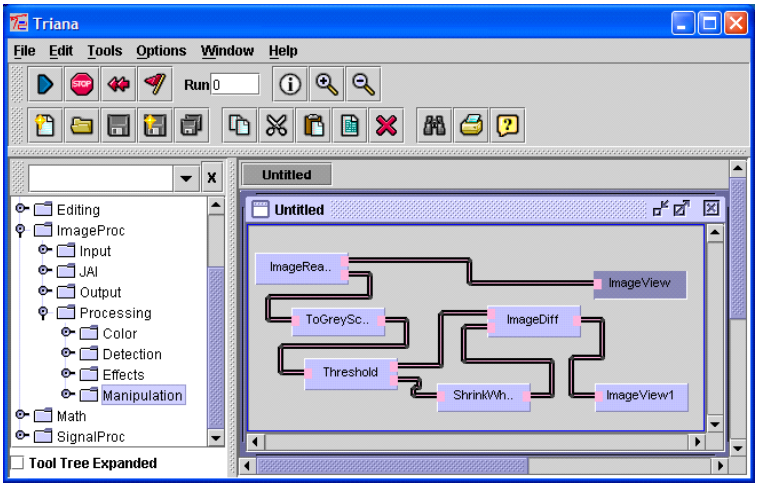

Figure 1: Triana being used to extract the edges from an image.

3 Triana Overview

Triana is an open source problem solving environment written in Java that enables the user to compose workflows from pre-written tools in a variety of problem domains. The Triana release includes tools for signal, image and text processing amongst others, and sophisticated data input and visualization tools. Triana also provides the facility for scientists to create their own tools using an easy to follow tool wizard, or through constructing group (composite) tools from a number of existing tools (see Section 3.1).

Workflows are composed by dragging the required tools from the tool tree onto the work surface and then wiring them together to create a workflow for the specific behaviour. For example, in Figure 1, we show Triana being used to compose a workflow to extract edges from an image. Triana has looping (do, while, repeat until etc.) and logic tools (if, then etc.) that can be used to graphically control the data-flow, just as a programmer would control the flow within a program by writing instructions directly. In this sense, Triana is a graphical programming environment.

3.1 Group Tools

Triana includes the ability for a user to create composite tools from a number of existing tools, for example a Power Spectrum can be created grouping the FFT, Magnitude and OneSide tools. In Triana we refer to these composite tools as group tools. Once created, a group tool behaves exactly like a singleton tool, in that it has a number of input and output nodes and can be connected into a workflow like any other tool. As well as allowing users to simply create powerful Triana tools, group tools play a key part in the workflow distribution within Triana, as described in Sections 4.1 and 4.2.

3.2 Triana Features

In addition to the distributed features introduced in the latest Triana release that we shall describe in Section 4, the following features are new or significantly improved:

User Interface - The Triana GUI conforms to more standard human-computer interface guidelines, and is thus more intuitive for first-time and experienced users.

XML Tool Definitions - Each Triana tool/group now has a standard XML definition. As there is no longer a one-to-one link between Java units and tools, several different tools based on the some unit code can be generated without replicating the underlying unit code (e.g. two FFT tools with different default parameters).

Pluggable Architecture - Triana now allows different workflow readers/writers to be plugged in, enabling Triana to be used as a graphical script editor for workflow composition, e.g. reading/writing WSFL and BPEL4WS formats. This feature was used in out recent collaboration with the GriPhyN project [Avery and Foster., 2001] to read/write VDLx/DAX workflows.

Customisable Tool Tree - The location of the tool XML files on disc and where they appear in the tool tree have been separated. This allows users to filter and customise their tool tree view to suit their requirements.

Tool Wizard - The tool wizard, including the tool GUI builder, has been rewritten to be more intuitive and flexible. The tool GUIs generated have a more standard look and feel.

4 Distributed Triana

A key feature of the new Triana release is that it enables users to distribute sections of a workflow to remote machines, and to connect their workflow to existing services already running on remote machines. As the users of Triana are unlikely to be familiar with Grid technologies such as OGSA and JXTA, a key challenge was to make distribution within Triana simple for non-grid familiar users understand and execute.

In the following section we look at how a user can distribute sections of their workflow to remote machines, and describe some of the underlying distribution mechanisms. In Section 4.2 we extend this to look a how sections of workflow can be distributed in parallel or pipelined. In Section 4.3 we look at how existing services on remote machines are discovered and connected into a user's local workflow. In these sections we generally look at the distribution from the user's perspective, ignoring the underlying Grid middleware; in Section 5 we describe our stop-gap middleware independent Grid interface that distributed Triana currently uses to distribute workflow on the Grid.

4.1 Distributing Workflow to Remote Services

The distribution of workflow using Triana first requires the existence of Triana launcher services running on remote machines. A launcher services provides the contact point for Triana on a remote machine, and facilitates the creation of actual Triana services executing workflow subsections. Basically, by sending a XML serialised workflow subsection to a launcher service located on a remote machine, Triana can initialises the workflow as a service running on that machine. The implementation of Triana launcher services is not specific to any architecture; so there can be an OGSA Triana launcher services for launching OGSA Triana services, a JXTA launcher services for launching JXTA Triana services and so on. Obviously, which service is contacted by Triana depends on its current underlying middleware (see Section 5).

Assuming the existence of remote Triana launcher services, from the users view distributing a task in the workflow to a remote machine is straightforward. A right-click on the task they wish to distribute brings up a pop-up menu with a menu option 'Distribute Properties'. Selecting this

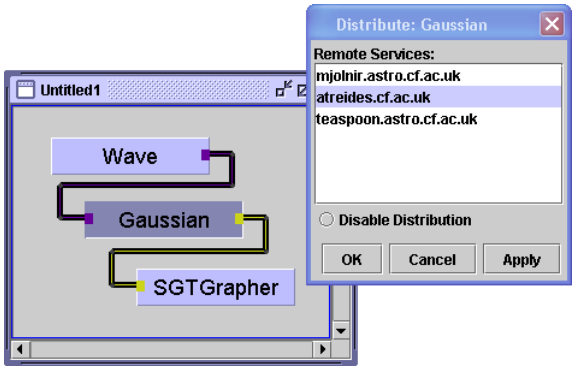

Figure 2: Selecting a remote service on which to distribute the Gaussian task.

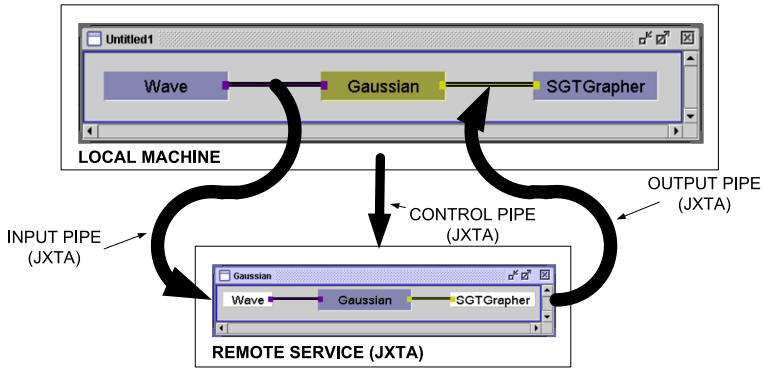

Figure 3: . The connection of a remote service into a Triana workflow using JXTA middleware.

option brings up a dialog prompting the user to select from the available machines (machines running launcher services), as shown in Figure 2. To distribute the task the user chooses the machine they wish to utilise, selects O.K., and Triana launches that task as a service running on the selected machine. As discussed in Section 3.1 multiple tools can be grouped to form group tools; to distribute a subsection of workflow as opposed to a single task simply involves the user grouping the subsection they wish to distribute, and distributing it as for a single task.

Obviously, although this process appears straightforward to the user, there are multiple stages behind the scene to create and connect to a remote service. The stages can be broken down into three steps:

Launch Service - Triana sends an XML serialisation of the workflow subsection to the Triana launcher service running on the remote machine. As described above, this causes a service providing that workflow to be created.

Connect to Service - Triana connects to the control pipe for the remote service, enabling it to send commands (e.g. run, stop) to the workflow and to steer the parameters of the tasks in that workflow.

Create Input/Output pipes - Triana connects input and output pipes to the nodes of the remote service, enabling data to be passed from the local workflow to the remote service, and the result to be passed back.

The protocol used by the control, input and output pipes again depends on the underlying middleware (see Section 5); if JXTA was being used then the pipes would be JXTA pipes (as

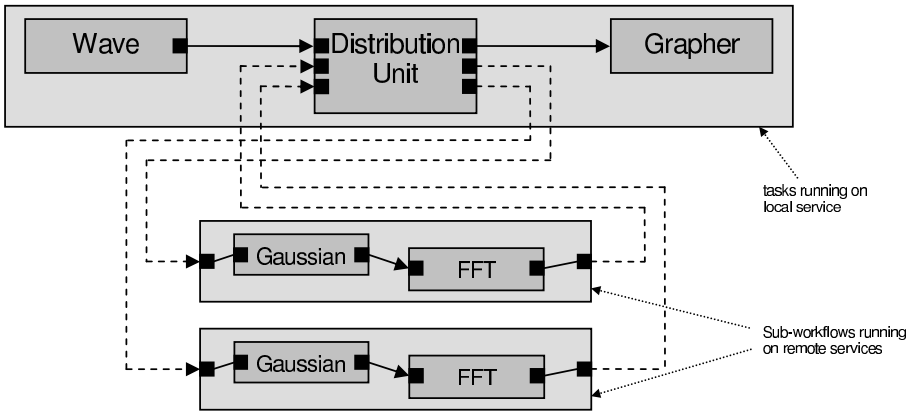

Figure 4: . The cloning and connection of sub-workflows using a parallel distribution unit.

illustrated in Figure 3), if web services was being used then it would be virtual pipes using HTTP and SOAP.

Once the workflow subsection has been distributed the user runs their workflow as standard and the data is automatically passed to/from the remote service via the input/output pipes. When the user quits their Triana session, they can leave the remote services they have created alive if they choose, allowing them to return to these workflows at a later date (e.g. when processing is complete), and also allowing others to utilise these services.

4.2 Custom Workflow Distribution

In the previous section we described the distribution of workflow to a single remote service. In this section we extend this to look at the custom distribution of workflow, for example the distribution of workflow in parallel or as a pipeline.

Custom distribution policies can only be applied to groups. The reason for this is that groups have a hidden control unit that handles the data passing into/out from the group (note that the control unit is standard Triana task). Normally the control unit handles looping, passing the data returning the output from a group back as its input until a certain condition is met. However, with custom distribution this control unit is replaced by a distribution unit. The distribution unit has two roles:

Workflow Annotation - the distribution unit cuts and clones the workflow within the group into sub-workflows. Each of these sub-workflows is distributed to a remote service for execution, and where required linked to/from the distribution unit, as shown for a parallel distribution in Figure 4.

Data Distribution - when data arrives at the distribution unit, the distribution unit decides which remote sub-workflow the data is sent to. For example, in a parallel distribution (see Figure 4), the data would be passed to which ever workflow had finished with its previous data item. The distribution unit is also responsible for regrouping data when it returns from the remote sub-workflows.

>From the users perspective custom distribution is again straightforward. When the user right clicks on the group and selects 'Enable Distribution' from the pop-up menu, they are offered a

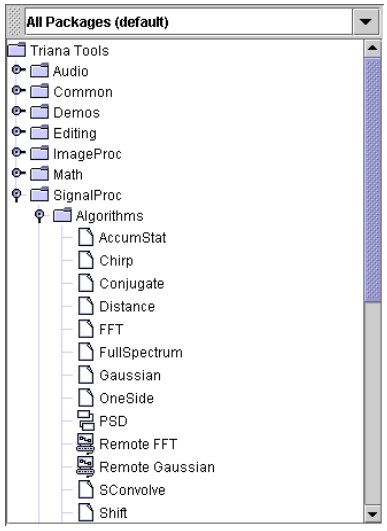

Figure 5: A Triana tool tree containing both remote tools (Remote FFT and Remote Gaussian) and local tools.

choice of using standard distribution (as described in 4.1) or using a custom distribution policy. If a custom distribution policy is chosen then the user interface for that distribution unit is shown; this enables the user to select which remote machines to utilise, or to allow Triana to automatically choose from the available machines (auto distribution).

4.3 Discovering and Connecting to Remote Services

In Triana remote services, either ones launched through Triana (see Section 4.1) or ones created by third parties, are considered remote tools that can be connected into a user's workflow in the same manner local tools. The only difference is that when the user visually connects a cable to a remote tool, this creates an actual pipe based using current grid-middleware (e.g. a JXTA pipe or a virtual HTTP pipe) rather than a local reference passing pipe.

The actual mechanism used to discover remote tools (services) depends on the underlying middleware (see Section 5), however, once a tool is discovered it is placed in Triana's tool tree along with the local tools. Remote tools therefore appear to the user almost identical to local tools, and the user does not need to concern themselves as to whether their tool is local, running on a JXTA peer or running as on OGSA service. The only indication to a user that a tool is remote is that it has a different icon in the tool tree, as shown in Figure 5. Note that if the user is only interested in remote tools then they can apply a standard filter to the tool tree that causes only remote tools to be displayed.

Like local tools, not only are remote tools displayed in the tool tree, but they can be dragged onto the work surface and connected into workflows, or, in the case of remote group tools, opened to inspect the contents. When a remote tool is dropped on the work surface user has the choice of attaching to the currently running service, launching the tool as a new remote service, or making a local copy of the tool (some tools may not offer all these options). It is worth mentioning here that some of these processes are still underdevelopment and not yet fully implemented at the time of writing.

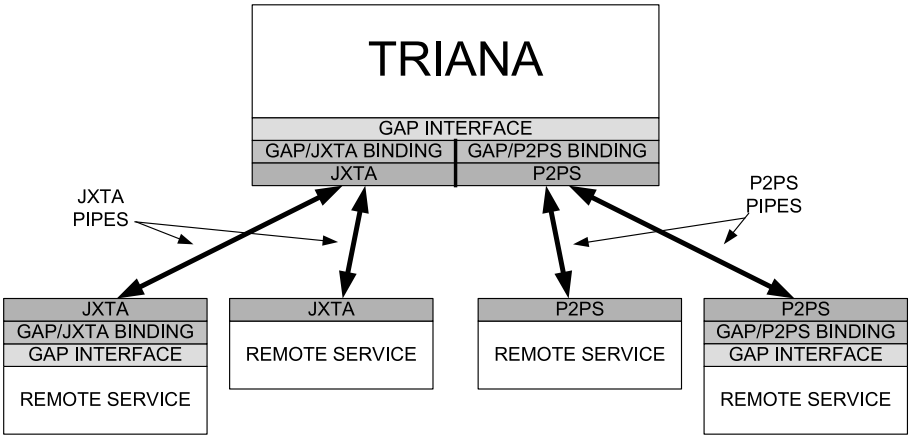

Figure 6: The distributed Triana architecture using the GAP Interface.

5 GAP Interface

The Grid Application Prototype Interface (GAP Interface) is a stop-gap grid API that we developed both as a suggestion to GridLab of the calls we require in the GridLab GAT-API, and as an interface for Triana to use while we await the GridLab GAT. The GAP Interface provides only a subset of the functionality that we expect to be in the GridLab GAT-API, but this is enough for the distribution mechanisms outlined in the previous sections to be implemented. The following is a incomplete outline of the calls in the GAP Interface:

Service Calls - advertiseService(servicename), locateServices (servicename), getServices(servicename).

Pipe Calls - createInputPipe(pipename), locatePipes(pipename), getPipes(pipename), connectOutputPipe(remotepipe)

Message Calls - send(message), addMessageListener(listener), removeMessageListener(listener)

Using the above calls Triana can create and locate remote services, connect input/output pipes to these remote services and send/received messages from them.

The GAP interface is not bound to any specific middleware. In fact we have currently have two GAP bindings, one for JXTA [http://www.jxta.org/, 2003], a set of protocols for decentralised peer-to-peer applications initially developed by Sun Microsystems, and one for P2PS, a simple socket based peer-to-peer toolkit. There are also plans to develop JINI and OGSA bindings.

In Figure 6 we illustrate the architecture of distributed Triana architecture using the GAP Interface. As can be seen from the diagram, different GAP bindings allow Triana to use services based of different middleware, for example the P2PS binding allows Triana to locate and connect to P2PS services. Also illustrated in the diagram, the GAP interface allows the location and connection to both services that were advertised themselves using the GAP Interface, and to services not using the GAP Interface.

When the GridLab GAT becomes available there are two options: the first is to write a GAP/GridLab GAT binding; and the second to replace the GAP Interface with the GridLab GAT. The first has the benefit that it is likely to be quicker to do as there should be a close correlation between the GAP Interface and the GridLab GAT-API, and that it leaves open the

option the carry on using the GAP/JXTA and GAP/P2PS bindings. However, the second option is the ultimate goal as that will allow Triana full access to the services provided by the GridLab GAT.

6 Summary

This document has outlined the release of Triana, as specified in Deliverable 3.4. In Section 2 we described how this release can be obtained from the Internet, while in later sections we summarised the basics of Triana and its distribution mechanisms. We have also described the GAP Interface, our GAT-API prototype, its current bindings and how it relates to the GridLab GAT-API.

References

[Allen et al., 2002] Allen, G., Angulo, D., Goodale, T., Kielmann, T., Merzky, A., Nabrzyski, J., Pukacki, J., Russell, M., Radke, T., Seidel, E., Shalf, J., and Taylor, I. (2002). Gridlab: Enabling applications on the grid. In *Proceedings of the 3rd International Workshop on Grid Computing (Grid2002)*.

[Avery and Foster., 2001] Avery, P. and Foster, I (2001). The GriPhyN Project: Towards Petascale Virtual Data Grids. http://www.jxta.org/.

[http://www.jxta.org/, 2003] http://www.jxta.org/ (2003). Jxta. http://www.jxta.org/.