# Programming Scientific and Distributed Workflow with Triana Services

Matthew Shields*
Schools of Physics and Astronomy and Computer Science,
Cardiff University

Ian Taylor
School of Computer Science,
Cardiff University

January 28, 2004

**Abstract**

In this paper, we give a status report of a real-world application scenario that uses two distinct types of workflow within the Triana problem solving environment: serial scientific workflow for the data processing of gravitational waves signals; and the distributed workflows that dynamically map to the virtual Grid overlay of Triana services. We briefly describe the Triana distribution mechanisms and the underlying architectures that we can support. Our middleware independent abstraction layer, called the GAP, enables us to advertise, discover and communicate with Web and P2P Triana Services. We then show how gravitational wave search algorithms have been implemented to distribute both the search computation and data across the European GridLab testbed, using a combination of Web Services, Globus interaction and P2P infrastructures.

## 1 Introduction

Triana is a graphical Problem Solving Environment (PSE), both a problem solving and a programming environment, providing a user portal to enable the composition of scientific applications. Users compose applications by dragging programming components, called units or tools, from toolboxes, and dropping them onto a scratch pad, or workspace. Connectivity between the units is achieved by drawing cables subject to type-checking. Although Triana was developed for use by data-analysis scientists in GEO 600 [5], it can be used in many different ways and around 500 tools exist covering a broad range of applications. Triana can be used as a Grid Computing Environment and can dynamically discover and choreograph distributed resources, such as Web Services, to greater extend its range of functionality. Triana has a highly decoupled modularized architecture [1] and each component can be used individually or collectively by both programmers, within their own applications, or end-users alike. [2].

*matthew.shields@astro.cf.ac.uk

## 2 Triana Workflow Language

A *component* in Triana is the unit of execution. It is the smallest granularity of work that can be executed and typically consists of a single algorithm or process. Components are Java classes with an identifying name, input and output "ports", a number of optional name/value parameters and a single *process* method. Components can also be written in other languages with appropriate wrapping code. Each component has a definition encoded in XML that specifies the name, input and output specifications and parameters the format is similar to WSDL [3] although slightly simpler. The definitions are used to represent instance information about a component within the workflow taskgraph language and component repositories. An example component definition can be seen below.

```
<tool>
   <name>Tangent</name>
   <description>Tangent of the input data</description>
   <inportnum>1</inportnum>
   <outportnum>1</outportnum>
   <input>
      <type>triana.types.GraphType</type>
      <type>triana.types.Const</type>
   </input>
   <output>...</output>
   <parameters>
      <param name="normPhaseReal" value="0.0"
          type="userAccessible"/>
      <param name="normPhaseImag" value="0.0"
          type="userAccessible"/>
      <param name="toolVersion" value="3"
          type="internal"/>
   </parameters>
</tool>
```

Triana is inherently flow based, it uses both data and control flow with component execution within Triana triggered by these flows. In the case of data flow, data arriving on the input "port" of the component triggers execution and in the case of control flow a control command triggers the execution of the component. Multiple inputs to a component can be set by the component designer to be mandatory, blocking until all received or optional, triggering immediately. The execution of workflow within Triana is decentralised, data or control flow "messages" being sent along communication "pipes" from *sender* to *receiver*. The communication can be either *synchronous* or *asynchronous* depending on the implementation of the communication pipe.

The internal workflow representation is object based, with specific Java objects for individual component instances or "tasks" and the hierarchy of connected tasks within a network. The representation is a Directed Cyclic Graph (DCG), *cyclic* connections are allowed within the Triana language, with *nodes* representing a component and *vertices* the connections between them. The external representation of the taskgraph is a simple XML syntax. A typical workflow consists of the individual participating component XML specifications and a list parent/child relationships representing the connections. Hierarchical groupings are allowed with sub-components consisting of a number of assembled components and connections. A simple example taskgraph consisting of just two components can be seen below.

```
<tool>
    <toolname>taskgraph</toolname>
    <tasks>
```

```
        <task>
            <toolname>Sqrt</toolname>
            <package>Math.Functions</package>
            <inportnum>1</inportnum>
            <outportnum>1</outportnum>
            <input>
                <type>triana.types.GraphType</type>
                <type>triana.types.Const</type>
            </input>
            <output>...</output>
            <parameters>
                <param name="popUpDescription">
                    <value>Square root of input data</value>
                </param>
                <param name="guiYPos" type="gui">
                    <value>76</value>
                </param>
                <param name="guiXPos" type="gui">...</param>
            </parameters>
        </task>
        <task>
            <toolname>Cosine</toolname>
            <package>Math.Functions</package>
            ....
        </task>
        <connections>
            <connection>
                <source taskname="Cosine" node="0" />
                <target taskname="Sqrt" node="0" />
            </connection>
        </connections>
    </tasks>
</tool>
```

Triana can use other external workflow language representations such as BPEL4WS [4] which are available through "pluggable" language readers and writers. Triana's execution engine uses the internal object representation and so the external representation is largely a matter of preference until a standards based workflow language has been agreed. Triana's XML language is not dissimilar to those used by other projects such as ICENI [6], Taverna/FreeFluo [7] and Ptollemy II [8] and should be interoperable.

A major difference between the Triana workflow language and other languages such as BPEL4WS is that our language has no explicit support for control constructs. Loops and execution branching in Triana are handled by specific components, i.e. we have a specific loop component that controls repeated execution over a sub-workflow and a logical component that controls workflow branching. We believe that this approach is both simpler and more flexible in that it allows for a finer grained degree of control over these constructs than can be achieved with a simple XML representation. Explicit support for constraint based loops, such as *while* or an optimisation loop, is often needed in scientific workflows but very difficult to represent. A more complicated programming language style representation would allow this but at the cost of ease of use considerations whereas our component based approach is both simple and extensible.

## 3 Workflow Distribution

This section briefly describes the underlying mechanisms that Triana uses in order to be able to distribute sections of its workflow and communicate with third party services e.g. Web Services. We introduce the concept of Triana virtual overlays that abstract the underlying middleware or transport bindings from the Triana programmer. Such, overlays are constructed through the use of the GAP interface, which is described in section 3.2. We then show how this is used within Triana to task-farm sections of Triana workflows.

### 3.1 Virtual Triana Overlays

Typically, current proposed solutions to both P2P and Grid computing involve the use of network overlays [10] [9] that attempt to abstract the underlying structure of the network from the programmers. Within Grid Computing, they employ the use of dynamic virtual organizations and OGSA services, whilst in Jxta, they employ the use of dynamic groups and peers to represent distributed resources. Within Triana we take this level of abstraction one step further through the use of the GAP interface, described in the next section. Triana distributed networks consist of an overlay of Triana GAP services that can be advertisted, discovered and communicated with using abstract high level calls that are independent of any underlying mechanisms that actually are used to distribute its behaviour on Grids and P2P networks. Such an overlay is at the GAP level and therefore can be employed by other applications as well as Triana.

Specific to Triana, constituting a layer above the GAP, is the GUI implementation within Triana which allows users to specify how they wish to distribute their Triana workflow. Within Triana, users, select code they wish to distribute by *grouping* sub-sections of the workflow. Such groups are simply collections of interacting units but are represented graphically by a single unit. A user can specify how their selected group is distributed by attaching a distribution policy to the group. There are currently two distribution policies: parallel, which is a task farming mechanism i.e. data parallel and generally involves no communication between hosted processes; and a pipeline mechanism, which involves distributing the group vertically, i.e. each unit in the group is distributed onto a separate resource and data is passed between them in a pipelined fashion. With the application described in this paper, we use the task-farming policy in order to distribute the inspiral search algorithm (itself tens of units) as a group to cooperating Triana GAP services across the Grid.

There are also two ways of distributing sections of workflows within Triana: dynamically, where Triana workflows are sent to generic Triana GAP services that can execute any sub-workflow and communicate with other Triana services they are connected to. This is analagous to RPC except that here whole collections of components are sent for execution on remote machines; or a user can chose to launch a group unit as a specific remote service so that it only thereafter provides this specify service. Using this method, all Triana units or groups may be deployed as Web Services.

### 3.2 Heterogeneous GAP Services

The Grid Application Prototype Interface (GAP Interface) is a generic high-level interface that provides a subset of the functionality of the GAT (Grid Application toolkit, created by GridLab [11]). The GAP is middleware independent but has bindings that adapt its capabilities to underlying middleware technologies. Currently, we have three bindings:

**Jxta** - Jxta [9] is a set of protocols for P2P discovery and communication within P2P networks.

**P2PS** - a lightweight P2P middleware capable of advertisement, discovery and virtual communication within ad-hoc P2P networks. P2PS has a subset of the functionality of Jxta but is tailored for simplicity, efficiency and is very stable.

**Web Services** - this allows applications to discover and interact with Web Services – using the UDDI registry [12] and the Web Service Invocation Framework (WSIF) [13]

The GAP is used to interface with Triana services and provides us with the middleware independent view of the underlying services and interactions across the Grid. The GAP is currently being extended to interact with Gridlab services, which in turn interface with Globus low-level services for job submission and data replication. It is also being extended to communicate directly with OGSA services so that, for example, a user using the GAP can easily choose between the Gridlab GRMS system or GRAM to implement their submission of a job.

## 3.3 Dynamically Distributing Triana Workflows

Triana provides a standard mechanism for distributing group tasks across multiple machines either in parallel or in a pipeline. Each group task is accompanied by a control task that receives the data input to the group before it is passed to the tasks within the group, and also receives the data output from the group before it is sent on from the group. Such control units dynamically rewire the workflow at run-time once they have information about the distribution policy and the number of services available.

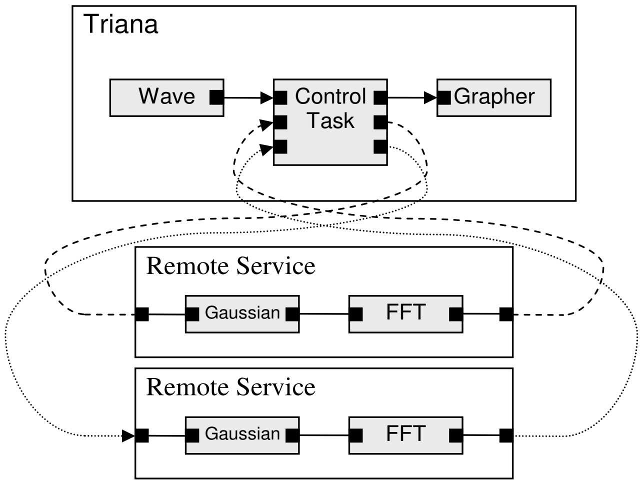

Figure 1: Triana dynamically rewires its workflow to connect to the remote services it has discovered through the GAP.

This is illustrated in Figure 1 which shows how the control unit would rewire the workflow in order to connect to two distributed services using the task-farming distribution policy. Here, you can see that the control unit actually reroutes the information from its input across to the various services running on the Grid and then awaits data before passing these to its output. For task-farming, the control unit simply feeds the available services with data passing new data to services that have completed previous analysis. The whole operation is asynchronous and the worker nodes (i.e. distributed services) are free to work at their own speed. Users can specify however, how the data should be output from the unit i.e. in the same order of arrival or as and when data is received from the worker nodes. Depending on their choice, the control units either buffer the data internally and the correct order is resumed or pass the data directly out when it is received, respectively.

## 4 Searching for Coalescing Binaries using Triana

This section gives an overview of how we have used a combination of the mechanisms described in this paper to implement a real-world scientific Triana example on the Grid. For example, the Web Services GAP binding is used to interact with the GridLab GRMS web service for job submission, the GAP P2PS binding is used to discover such services and a combination of GAP web service invocation and low level library interfaces are used to connect to the GridLab data management system. In the next section, we briefly describe the scientific goals of the scenario, then we describe how this was implemented in Triana and how this is mapped onto the testbed using the various services.

### 4.1 Inspiral Search Algorithm Triana Implementation

Einstein's theory of General Relativity predicts the existence of gravitational waves. Here, we describe how Triana has been applied to search for waves that are generated by compact binary stars orbiting each other until their ultimate collision. Laser interferometric detectors such as GEO600, LIGO and VIRGO should be able to detect the waves from the last few minutes before collision. To search for an inspiral signal (or coalescing binary), thousands of templates, representing the theoretical knowledge of relativistic binary systems, are correlated with the signal using fast correlation by a technique known as *matched filtering*. Each template contains different parameters defined at a certain granularity within the search space.

The implementation within Triana [14] uses approximately fifty separate algorithmic components connected together to form a workflow (shown in the top left hand section of Figure 2). Here, we see Triana used as a graphical programming tool that reuses much existing code in order to generate new methods for analysing data. The application scientist can easily try new novel detection methods by inserting or replacing units to and from the workflow. This workflow is the first type, i.e. serial scientific workflow representing an algorithm.

### 4.2 Mapping the Search onto the Gridlab Testbed

Triana is currently being applied to implement this inspiral search workflow on a number of resources distributed across Europe within the GridLab testbed. There are a number of extensions that have been/are being built within Triana in order to achieve a production Grid for this application. Firstly, we have extended our Web Services implementation to integrate the Grid Security Infrastructure (GSI) based on X.509 certificates into the GAP. This allows us to contact the secure GridLab services that are currently available to us and deployed on the GridLab testbed. There are two services that we are presently interested in:

**Job Submission** - using GSI we will be able to use GridLab services from within a Triana workflow in the same way that we can invoke Web Services now. This will not only allow us to connect to the GridLab GRMS service (for Globus-based job submission) but it will allow us to choreograph job submission workflow for complex submissions e.g. job submission could involve a number of steps: CVS checkout, compilation, service deployment etc. Figure 2 shows this use of the second type of workflow within Triana. Here Triana is using GRMS

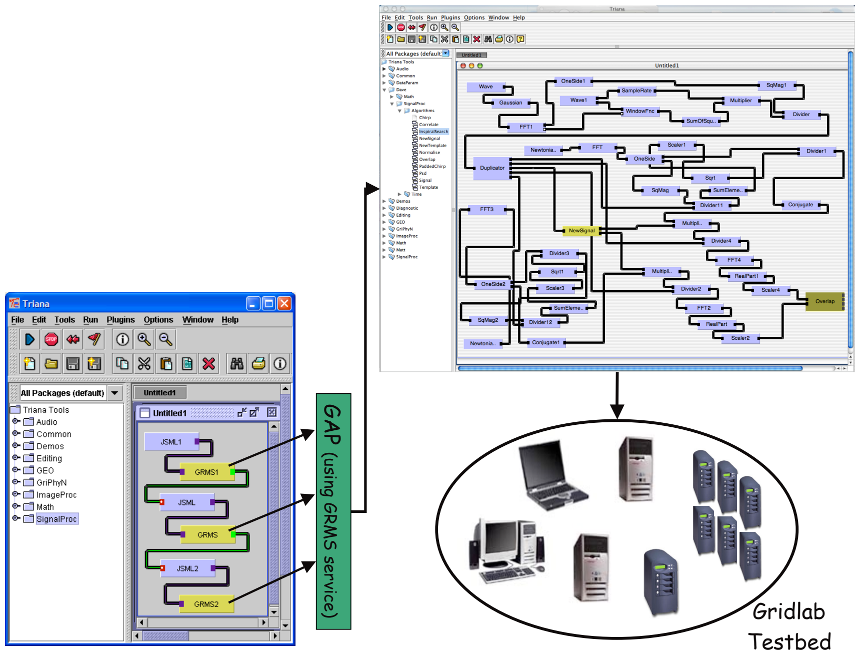

Figure 2: A workflow for the job submission of Triana services.

to start Triana Services on the testbed that will then be used to run multiple copies of the inspiral search algorithm. Effectively Triana is submitting instances of itself to run on the Grid.

**Data Management** - the detector data will be stored in a decentralised fashion across the GridLab testbed and retrieved through the GridLab data management services. The data management team have already replaced the communication layer of the gravitational wave IO library for reading/writing data (the Frame Library), allowing geographical transparency by treating local and remote access identically using a logical file reference. The user provides a GPS time as the logical filename, which is converted into the file location on the set of distributed resources.

Figure 3 shows the interaction between the client and the various Triana services running on the testbed. The workflow on the client is relatively simple, containing: an input unit that specifies the GPS second of the data to be loaded by one of the worker nodes along with the data length; a processing group node (containing the algorithm); and an output node that post processes the results returned from the workers. The results contain a minimal amount of data, e.g. the GPS second and the correlation ratio of the detected binary, and are only sent if something interesting has been detected. Such events, typically of the order of a few per year are a communicatively trivial but important step. On the client side, we are considering the use of various notification schemes upon successful detection e.g. email notifiation, SMS notification and screen alerts, which are readily available as Triana components.

The client distributes the group containing the binary search algorithm to all available nodes on the testbed. On the GUI the user right clicks the group bringing up a menu to enable its

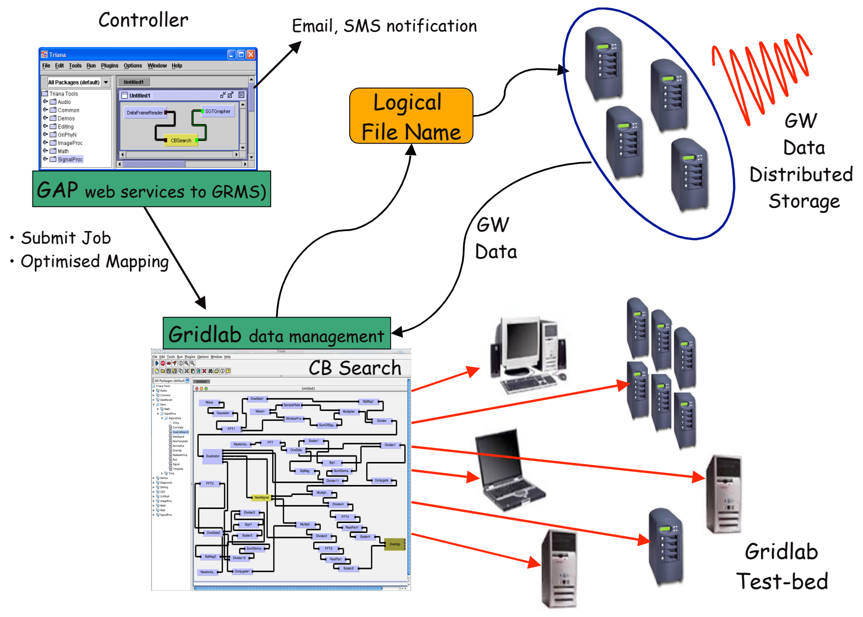

Figure 3: A illustration of the task-farming of the inspiral search algorithm and the interaction with the GAP and the Gridlab services.

distribution, then they specify the distribution policy for the group e.g. parallel, to task-farm the code across the resources. Triana then uses the GAP to locate the available Triana services (that have been launched using our interface with the GRMS service) and set up connections using the GAP control pipes. The client passes the workflow to each service to convert them into binary search worker nodes on the testbed. Once set up the services are ready for use and the algorithm proceeds in the following way:

**Initialisation and Data Retrieval** - a logical file name is transmitted to a node, which uses the data management service to return the physical file location that is used by the remote frame library to access the data.

**Processing** - the node initialises i.e. generates its templates (a trivial computational step) and then runs the algorithm performing fast correlation on the data set with each template in a library of around 10,000 templates.

**Results** - Results are returned back to the client from any workers that have detected something interesting.

## 5 Conclusion

In this status report we have outlined the current implementation of a real-world, data and compute intensive, scientific application on a Grid testbed. We use the Triana PSE as both a visual programing tool to implement a specific algorithm, and as a job submission system to run the implemented algorithm in a data parallel fashion on available resources on the testbed. We utilise

GSI enabled Web Services from GridLab, for job submission and data management, together with P2P technologies for communicating between the running Triana Services. Triana uses our GAP interface, a subset of the GridLab GAT, as an abstraction from the underlying middleware technology. This work shows that Triana is both a powerful visual programming tool and an enabler for Grid technology, providing a number of “ease of use” benefits to the scientist not interested computer science implementations.

## 6 Acknowledgements

The authors would like to thank the other members of the group for their invaluable contributions to the demonstration application discussed in this paper. The astronomers: Prof. B Sathyaprakash, David Churches, Roger Philp and Craig Robinson; the Triana team: Ian Wang, Andrew Harrison, Omer Rana, Diem Lam and Shalil Majithia; and all the partners in the GridLab project.

## References

[1] Ian Taylor, Matthew Shields and Ian Wang, Book, Grid Resource Management, edited by Jan Weglarz, Jarek Nabrzyski, Jennifer Schopf and Maciej Stroinski, Kluwer, June 2003.

[2] Ian Taylor, Matthew Shields, Ian Wang and Roger Philp Grid Enabling Applications Using Triana Workshop on Grid Applications and Programming Tools, June 25, 2003, Seattle. In conjunction with GGF8 jointly organized by: GGF Applications and Testbeds Research Group (APPS-RG) and GGF User Program Development Tools Research Group (UPDT-RG)

[3] W3C Web Services Description Language (WSDL) 1.1 W3C Note, March 15, 2001. see website *http://www.w3.org/TR/wsdl*

[4] BPEL4WS. ”Business Process Execution Language for Web Services”, Version 1.1 05 May 2003. See *http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/*

[5] GEO 600 Gravitational Wave Project Home Page, see *http://www.geo600.uni-hannover.de/*

[6] London e-Science Centre ICENI Project Home Page, see *http://www.lesc.ic.ac.uk/iceni/*

[7] The Taverna Project Home Page, see *http://taverna.sourceforge.net/*

[8] The Ptolemy II Project Home Page, see *http://ptolemy.eecs.berkeley.edu/ptolemyII/*

[9] Project JXTA, see website *http://www.jxta.org/*

[10] Foster, I., and C. Kesselman, eds. The Grid: Blueprint for a New Computing Infrastructure Morgan-Kaufmann, 1999.

[11] Gabrielle Allen, Kelly Davis, Konstantinos N. Dolkas, Nikolaos D. Doulamis, Tom Goodale, Thilo Kielmann, André Merzky, Jarek Nabrzyski, Juliusz Pukacki, Thomas Radke, Michael Russell, Ed Seidel, John Shalf and Ian Taylor. Enabling Applications on the Grid: A GridLab Overview, International Journal of High Performance Computing Applications: Special issue on Grid Computing: Infrastructure and Applications, August 2003.

[12] UDDI.org UDDI Technical White Paper UDDI.org, September 6, 2000. see website *http://www.uddi.org*

[13] Web Services Invocation Framework (WSIF), see website *http://ws.apache.org/wsif/*

[14] David Churches, Matthew Shields, Ian Taylor and Ian Wang. A Parallel Implementation of the Inspiral Search Algorithm using Triana. Proc. of UK eScience All Hands Meeting Sept. 2-4, 2003, Nottingham.