IST-2001-32133
GridLab - A Grid Application Toolkit and Testbed

D3.5 Heterogeneous Distributed Runs

Author(s):	Dave Churches, Alex Gray, Andrew Harrison, Shalil Majithia, Roger Philp, Omer Rana, Craig Robinson, B. Sathyaprakash, Bernard Schutz, Matt Shields, Ian Taylor, David Walker and Ian Wang

Document Filename:	GridLab-3-D3_5-0001-HeterogeneousRuns
Work package:	WP3 Work-Flow Application Toolkit (TGAT)
Partner(s):	University of Wales, Cardiff
Lead Partner:	University of Wales, Cardiff
Config ID:	GridLab-3-D3_5-0001-1.0
Document classification:	INTERNAL

Abstract: In this document, we describe the heterogeneous Triana implementation that can work on multiple platforms and use a variety of middleware types in order to distribute sections of the workflow. We briefly describe the Triana distribution mechanisms and through the independent abstraction layer, called the GAP (a subset of the GAT), that enables us to advertise, discover and communicate with Web and P2P Triana Services. We then give a status report of how we are planning to map gravitational wave search algorithms onto the GridLab testbed, using a combination of Web Services, Globus interaction and P2P infrastructures.

Contents

1 Introduction

Triana is a graphical Problem Solving Environment (PSE) that enables scientists to graphically program applications. Users write programs by dragging 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 modularised architecture [1] and each component can be used individually or collectively by both programmers, within their own applications, or end-users alike. [2].

2 Heterogeneous Component Distribution

This section describes the underlying Triana mechanisms in order to distribute sections of its workflow to a collection of heterogeneous devices and communicate with third party services e.g. Web Services. Triana applies a virtual overlay, called the GAP, that abstracts the underlying middleware or transport bindings from the Triana programmer. We show how this is used within Triana to task-farm sections of Triana workflows.

2.1 Virtual Triana Overlays

Typically, current proposed solutions to both P2P and Grid computing involve the use of network overlays [7] [6] that attempt to abstract the underlying structure of the internet from the programmers. In Grid Computing, dynamic virtual organisations and OGSA services are employed, whilst in P2P systems e.g. Jxta, they employ the use of dynamic grouping and peer protocols to represented distributed resources. Within GridLab 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 advertised, discovered and communicated by using abstract high level calls that are independent of any underlying distributed mechanisms that bind it to Grids or 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. Briefly, users select code they wish to distribute by *grouping* sub-sections of the workflow, which are thereafter represented graphically by a single unit. A group can then be distributed by attaching a distribution policy specifying the distributed behaviour. There are 2 types: parallel i.e. task-farming or SIMD applications that 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. For the Inspiral algorithm 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 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 analogous to RPC except that here whole collections of components are sent for execution on remote machines

- **statically:** by pre-launching a group unit as a specific remote service so that it only thereafter provides this specific service. Using this method, all Triana units or groups may be deployed as Web Services.

2.2 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 interface [8]. The GAP is a lightweight Java implementation that provides a similar level of abstraction to the GAT but with limited functionality. It is middleware independent, with bindings provided for different Grid middleware such as JXTA and Web Services. One key focus for the GAP is to support Triana P2P interactions on the "Consumer Grid" [14] for running massively parallel high-throughput codes.
Part of the motivation behind the GAP Interface is as a stopgap to enable us to develop distribution mechanisms within Triana while the GridLab GAT is being developed. When the GridLab GAT becomes available the GAT-API will replace the GAP Interface within Triana and should enable Triana to make use of the advanced security, logging and other GridLab services. However, the GAP Interface will live on, both as a simple interface for prototyping Grid and P2P applications, and as an adaptor within the GridLab GAT architecture providing various discovery and communication capabilities. Currently there are three GAP bindings implemented:

JXTA - The original GAP Interface binding was to JXTA [6]. JXTA is a set of protocols for Peer-to-Peer discovery and communication originally developed by Sun Microsystems. Although we achieved some initial success with JXTA, we have since had problems with the speed and reliability of our JXTA binding.

P2PS - a lightweight Peer-to-Peer middleware we have developed in Cardiff. P2PS (P2P Simplified) is a lightweight P2P infrastructure based on XML advertisements and messaging. As the name suggests, P2PS aims to provide a simple infrastructure on which to develop P2P style applications, without the complexity of other similar architectures such as JXTA and Jini [12]. Like JXTA, the P2PS infrastructure employs XML in its discovery and communication protocols, and is independent of any implementation language or computing hardware. Assuming that suitable P2PS implementations exist, it should be possible to form a P2PS network that includes everything from super-computer peers to PDA peers. Furthermore, communication within P2PS is not tied to any single transport protocol, such as TCP, and can be extended to include new protocols, such as Bluetooth. The current reference implementation of P2PS is written in Java, and handles pipe communication over both TCP and UDP by default. Although P2PS is not an implementation of the JXTA protocols, its architecture is inspired by that of JXTA. However, P2PS focuses only on the core elements required for peer discovery and pipe-based communication.

Web Services - The most recent GAP binding allows applications to discover and interact with Web Services. In this case, service descriptions are specified as a Web Service Description Language (WSDL)document, and can be published into a UDDI registry [9]. When a WSDL description of a Web Service is discovered or imported, it is parsed by Triana and a Triana tool representing each of the operations available on the service is generated. The input and output nodes on each tool represent the input message parts required by the Web Service and the message parts returned from the Web Service operation respectively; for example, if a Web Service operation requires two input message parts then the generated tool has two input nodes. The Triana units generated from discovered/imported WSDL

documents are inserted by Triana into the user's tool tree, alongside the available local tools. These Web Services can thereafter be used as standard Triana units. Invocation of Web Services is currently done using the Web Services Invocation Framework [13]. When data is sent from a local Triana unit to a Web Service, the data from each input cable for that service is packaged into a WSIF input message, and any data type conversions (e.g. string to double) are achieved. The Web Service is then invoked with the input message, and the data returned by the Web Service is passed to the next tool in the workflow along the relevant output cable. If the next tool is a Web Service then the return data is used to invoke that service, allowing Triana to choreograph workflows involving multiple Web Services. This feature combined with the BPEL4WS reader that we are developing will allow us to import and choreograph BPEL4WS workflows.

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.

2.3 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.

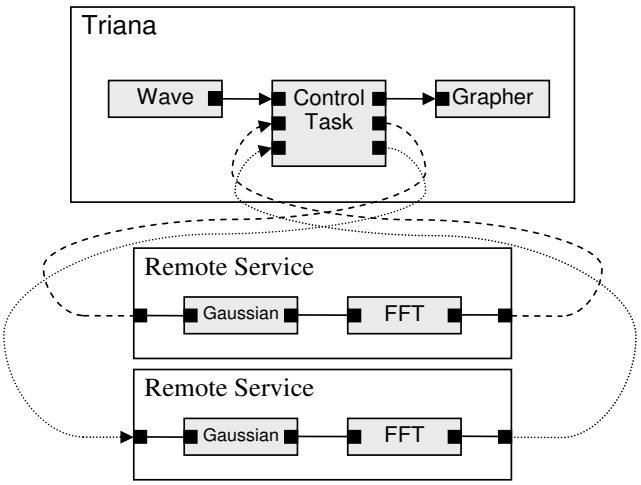

Figure 1: Triana dynamically rewires its workflow to connect to the remote services it has discovered through the GAP.

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 passes the data directly out when it is received, respectively.
Briefly, when Triana is started it automatically uses the GAP Interface to search for existing published Triana GAP services, and any that are found are inserted into the users tool tree alongside the existing local tools. Additionally, a "Discover Services" option may be used to issue a GAP discover services call. When a remote service is located it is inserted into the users tool tree. Once discovered, remote tools can be dragged and connected into the users workflow in exactly the same manner as local tools. However, when an input/output cable is connected to a remote task, Triana connects an input/output GAP pipe to the remote service instead of a local cable.
A key feature is the capability to advertise workflow subsections (if authorised), launched as remote Triana services, using the GAP bindings (e.g. P2PS advertisement for the P2PS/GAP binding or UDDI for the Web Services/GAP binding. Once advertised these Triana services can be discovered and invoked by both other instances of Triana and by non-Triana related applications. This is a powerful feature as Triana can be used to quickly create and launch a wide range of composed Triana algorithms as services.
In the next section, we give a status report on how we are using a combination of these mechanisms to implement a real-world scientific Triana application on the Grid.

3 Searching for Coalescing Binaries using Triana

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.
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.
This integration has involved developing or integrating the following features:

- **Core Algorithm Development:** The inspiral search implementation within Triana [11] 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.

- **Web Services Integration**: we use the Web services GAP binding to interact with the GridLab GRMS Web service for job submission and to the GridLab Data Management

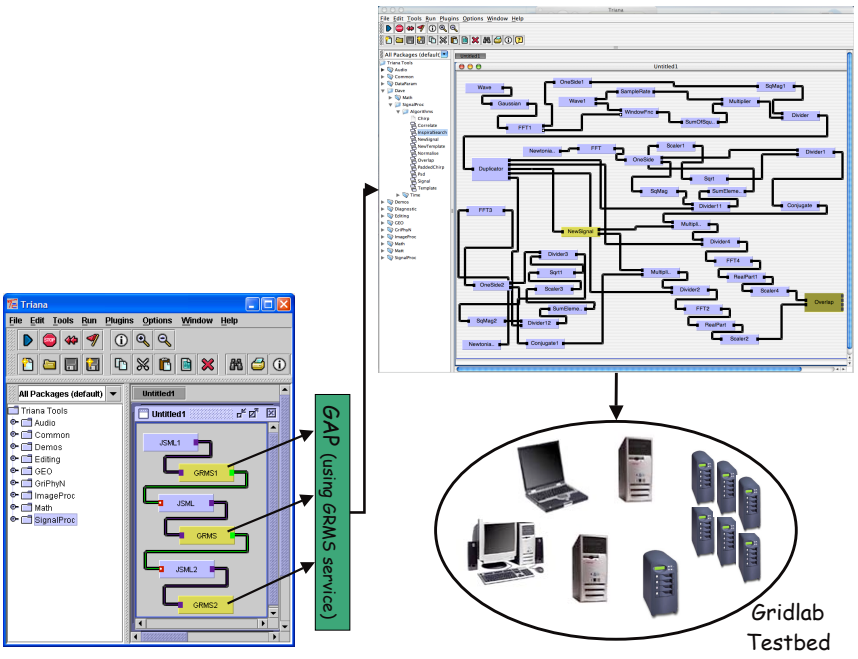

Figure 2: A workflow for the job submission of Triana services.

service for converting the logical filename (a GPS time) into a URI.

- **Secure GAP Web Services**: 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.

- **Job Submission** - using GSI, we can 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.

- **Data Library Integration:** the Triana *FrameReader* unit can load in a gravitational-wave data file, which can consists of multiple seconds worth of data from a large number of channels. For example, in GEO 600 [5], 60 second Frame files constituting over a hundred slow and fast data channels. The Data Management team integrated the Frame C library with their data management system in order to be able to load Frame files remotely. The Triana team have then modified their FrameReader unit to use this library and to allow the user to provide a logical filename instead of an actual one. Therefore, the detector data will be stored in a decentralised fashion across the GridLab testbed and retrieved through these GridLab data management services, 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.

- **P2P Interaction:** the GAP P2PS binding is used to discover the launched Triana GAP services and we use our abstract GAP Pipe interface for communication.
- **P2PS:** We use our P2PS binding for the underlying mapping of our virtual Triana GAP services onto the Grid. P2PS implements several discovery mechanisms and communication via TCP and UDP sockets for discovering and communicating with the remote services. Note that such an overlay is only an overseer for coordinating the distributed workflow. The heavyweight data transfers are provided by the Grid data management services directly. This is described in detail in the following section.

3.1 The Inspiral Search Application Scenario Illustrated

Figure 3 shows this 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 notification, SMS notification and screen alerts, which are readily available as Triana components.

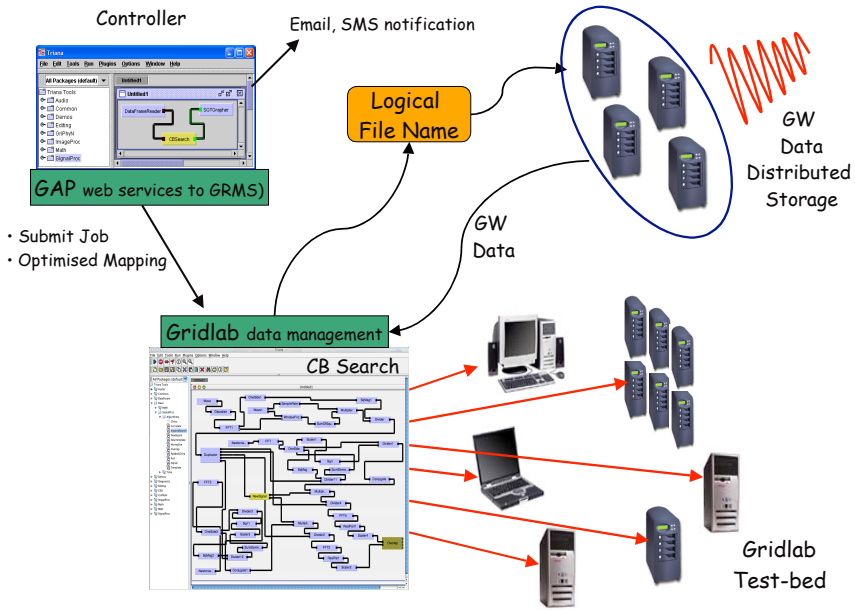

Figure 3: An illustration of the task-farming of the inspiral search algorithm and the interaction with the GAP and the Gridlab services.

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

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.

4 Conclusion

In this deliverable we have outlined how Triana enables distributed across a collection of heterogeneous devices using the GAP interface. The GAP provides Triana with a virtual overlay network that can be used to abstract the underlying middleware mechanisms away from the Triana programmer and allows true interoperability between a diverse set of services. We have also shown how Triana could submit itself as a "job" to the GridLab testbed and how it can be used to implement an algorithm in a data parallel fashion across available resources. 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.

5 Acknowledgements

The authors would like to the various partners in the GridLab project for their programming effort and help in providing access to their GridLab services, especially the Data Management and Resource Management Work Package Groups. We also acknowledge effort from Diem Lam within the Web services integration within his MSc 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] Project JXTA, see website *http://www.jxta.org/*

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

[8] 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.

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

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

[11] 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.

[12] Jini, see website *http://www.jini.org/*

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

[14] I. J. Taylor, R. Philp, O. F. Rana, M. Shields, and I. Wang. Supporting Peer-2-Peer Interactions in the Consumer Grid. Proceedings of HIPS Workshop at IPDPS, April 2003.