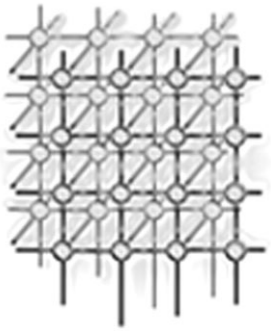
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2006; **18**:1021–1037
Published online 13 December 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.992



Programming scientific and distributed workflow with Triana services

David Churches[1], Gabor Gombas[2], Andrew Harrison[3], Jason Maassen[4], Craig Robinson[1,3], Matthew Shields[1,3,*,†], Ian Taylor[3] and Ian Wang[1,3]

[1]*School of Physics and Astronomy, Cardiff University, Cardiff, U.K.*
[2]*Laboratory of Parallel and Distributed Systems, MTA SZTAKI, Computer and Automation Research Institute of the Hungarian Academy of Sciences, Budapest, Hungary*
[3]*School of Computer Science, Cardiff University, Cardiff, U.K.*
[4]*Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands*

SUMMARY

In this paper, we discuss a real-world application scenario that uses three distinct types of workflow within the Triana problem-solving environment: serial scientific workflow for the data processing of gravitational wave signals; job submission workflows that execute Triana services on a testbed; and monitoring workflows that examine and modify the behaviour of the executing application. We briefly describe the Triana distribution mechanisms and the underlying architectures that we can support. Our middleware independent abstraction layer, called the Grid Application Prototype (GAP), enables us to advertise, discover and communicate with Web and peer-to-peer (P2P) services. We 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. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: Triana; distributed systems; Grid; workflow; peer-to-peer

1. INTRODUCTION

Workflow has been used in business activities for a number of years, in areas as diverse as manufacturing production line systems through to business process management, to manage consistency over repeated procedures. In scientific and engineering computing, workflow has been adopted more recently for similar reasons. Workflow can be used to control the interaction of

*Correspondence to: Matthew Shields, School of Physics and Astronomy, Cardiff University, 5 The Parade, Cardiff CF24 3YB, U.K.
†E-mail: matthew.shields@astro.cf.ac.uk

Received 17 July 2004
Revised 11 March 2005
Accepted 27 April 2005

discrete processes or computations and provide a mechanism by which those interactions can be represented and reproduced. With the advent of service-based computing, especially Grid services [1] and Web services, there is an increased interest in a layer of control above the service level that can be used to choreograph interactions between services. Mechanisms are required for representing dependencies between services, either temporal or data driven dependencies; controlling constructs, such as conditional branching or loops; and scheduling/execution of completed workflows. Workflow languages and workflow systems such as Problem Solving Environments (PSEs) and Grid Computing Environments (GCEs) can provide the means to accomplish some or all of these tasks.

This paper provides a brief overview of the Triana PSE and the workflow representations it uses to manage interacting Triana services. We define three distinct types of workflow: serial scientific workflow for data processing; job submission workflows that deploy services on a testbed; and monitoring workflows that examine and modify the behaviour of the executing application. We conclude with a real-world example of a scientific application running on the European GridLab testbed.

Section 2 provides a brief overview of Triana and its workflow language, and compares it to other similar systems and languages. Section 3 discusses how workflows are distributed using Triana and how communication is performed with external services, e.g. Web services. This section includes descriptions of the Grid Application Prototype (GAP) and Grid Application Toolkit (GAT) [2], and the various bindings for those interfaces: JXTA, P2PS, and Web services. A mechanism for supporting task farming or SPMD[‡] parallelism is also outlined. A real-world application scenario is presented in Section 4. A brief discussion of the application implementation and the problem domain is given, followed by a description of how the application is mapped onto the GridLab Computational Grid Testbed. Included in this section is a discussion of job submission on the testbed using the Grid(Lab) Resource Management System (GRMS), and data management through the GridLab data management services. This section concludes with an examination of dynamic resource allocation for the application on the testbed, and how this is achieved though dynamic application monitoring and adapting to quality of service constraints.

2. TRIANA, WORKFLOW AND RELATED WORK

Triana is a graphical PSE, providing a user portal for the composition of scientific applications. Users compose applications by dragging programming components, called units or tools, from toolboxes and dropping them onto a 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 [4], it can be used in many different ways and around 500 tools currently exist covering a broad range of applications. An overview of Triana's operation can be found in [5].

A recent survey by Fox *et al.* [6], as part of the Global Grid Forum's (GGF) GCE working group, concluded that PSEs generally provide 'some back-end computational resources, and convenient access

[‡]SPMD (adj.) represents single program, multiple data; a category sometimes added to Flynn's taxonomy to describe programs made up of many instances of a single type of process, each executing the same code independently. The HPCC Glossary is given in [3].

to their capabilities'. Workflow features significantly in many of the descriptions in this survey, and in many cases, access to data resources is provided in a similar manner to computational resources. Often the terms PSE and GCE are used interchangeably, as PSE research predates the existence of Grid infrastructure.

Triana can be used as a GCE and can dynamically discover and choreograph distributed resources, such as Web services, to extend its range of functionality. Triana has a highly decoupled modularized architecture [7] that allows components to be used individually or collectively by workflow programmers and end-users alike [8].

Triana like many contemporary systems uses an XML language to represent both component definitions and workflow. Most component definitions have a similar structure and contain definitions for component name and identification plus input and output port specifications for the type and number of input and output data items. One well known XML component definition is WSDL [9], an interface definition language for describing Web services.

SCIRun [10], ICENI [11], Taverna/FreeFluo [12], and Kepler [13] are all PSEs that use similar XML-based component models. SCIRun is a PSE for parallel and scientific computing and medical modelling in particular, like Triana it has an XML component model and uses data flow within directed graphs as its execution model. ICENI, Taverna and Kepler are very similar PSEs in intent to Triana. Each has a component and workflow model implemented in XML. ICENI uses larger grained components than the others, choosing to focus on large Grid-enabled applications as components. Taverna originally started life as purely a Web services choreography tool for performing computational biology experiments but now includes support for Java-based components as well. The focus of Kepler is actors, where an actor is a re-usable component that communicates with other actors through channels. Kepler works in two domains: either a process network where actors model a series of processes, communicating by messages through channels; or in a synchronous data flow domain where the communication along the channels is the data flow. The Fraunhofer Research Grid [14] are developing a component-based PSE with an XML component model that uses petri nets as opposed to the Directed Acyclic Graphs (DAGs) used by many of the other systems.

XCAT [15] is a project to implement the Common Component Architecture (CCA), a proposed standard for portable software components, and compliant frameworks for executing them, onto a Grid infrastructure. It would provide the framework upon which a workflow system could be built.

The Work Flow Management research group (WFM-RG) in the GGF is currently exploring application workflows and their execution in a Grid environment. An ongoing survey of scientific workflows [16] is identifying the current active workflow research projects and the models and languages they use. One of the goals of the WFM-RG is to get a consensus on workflow standards. Current proposed standards for workflow include the Business Process Language for Web Services (BPEL4WS) [17] and Web Services Flow Language (WSFL) [18], both of which are workflow languages specifically designed for use with Web services. Syntactically they are very similar to the workflow language Triana uses with specifications for components or services, and the connections between them. DAGMan [19] is a workflow scheduling system for the Condor scheduling system, using workflows specified as directed acyclic graphs. The syntax is not XML based and the dependencies have to be acyclic unlike Triana which allows cyclic dependencies.

A major difference between the Triana workflow language and other languages such as BPEL4WS is that it has no explicit support for control constructs. Loops and execution branching in Triana are handled by specific components. 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 optimization 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. The component-based approach discussed here is both simple and extensible without the need for extensive component language extensions.

2.1. Components and workflow

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 although slightly simpler. The definitions are used to represent instance information about a component within the workflow 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" type="userAccessible">
      <value>0.0</value>
    </param>
    <param name="toolVersion" type="internal">
      <value>3</value>
    </param>
  </parameters>
</tool>
```

Triana uses both data and control flow. In the case of data flow, data arriving on the input ports of the component trigger execution. In the case of control flow, a control command trigger the execution of the component. Inputs to a component can be set by the component designer to be mandatory, blocking execution until data are received, or optional, triggering immediately. The execution of workflow within Triana is decentralized, with data or control flow messages being sent along communication pipes without returning to a central point of control. The communication can be either *synchronous* or *asynchronous* depending on the implementation of the communication pipe.

Triana's internal workflow representation is object based, with specific Java objects for individual component instances (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 *edges* 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 of 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, which are available through pluggable language readers and writers. Triana's execution engine uses the internal object representation, so the external representation is largely a matter of preference until a standards-based workflow language has been agreed.

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 virtual Grid overlays of Triana services that abstract the underlying middleware and 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 Grid overlays of Triana services

Typically, current proposed solutions to both P2P and Grid computing involve the use of network overlays [20,21] 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 and P2P computing, 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, a high-level programming interface for service advertisement, discovery and communication. Triana distributed networks consist of an overlay of Triana GAP services that can be advertised, discovered and communicated with by using abstract high-level calls that are independent of the underlying mechanisms used to distribute the behaviour onto Grids and P2P networks. Such an overlay is at the GAP level, below Triana itself, 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 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 compound unit. A user can specify how their selected group is distributed by applying a distribution policy to the group. There are currently two distribution policies:

parallel: a task farming mechanism—data parallel model with no communication between hosted processes;

pipeline: a vertically distributed model—each component in the group is distributed onto a separate resource and data are passed between them in a pipelined fashion.

A distribution policy is a workflow rewriting mechanism that takes the original workflow and recombines it in new ways. For instance, in the *parallel* or task-farming policy the distributed sections of the workflow are duplicated and reconnected within the workflow to implement the SPMD mechanism on available Triana services. With the application described in this paper, we use the

task-farming policy to distribute the inspiral search algorithm, implemented as a group of units, to cooperating Triana GAP services across a computational Grid testbed.

There are two mechanisms for executing distributed sections of workflows within Triana: a static approach and a dynamic approach. In the dynamic approach, 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. In the static approach, a user can chose to launch a group unit as a specific remote service with a fixed interface so that it only thereafter provides this specific service. Using this method, Triana units or groups may be deployed as Web services.

3.2. Heterogeneous GAP services

The GAP Interface is a generic high-level interface that provides a subset of the functionality of the GridLab GAT. The GAP is middleware independent but has bindings that adapt its capabilities to underlying middleware technologies. Currently, we have three bindings.

JXTA [21] is a set of protocols for P2P discovery and communication within P2P networks.

P2PS is 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 stability.

Web services allows applications to discover and interact with Web services using the UDDI registry [22] and the Axis SOAP Engine [23].

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, Grid Resource Allocation and Management (GRAM) [24], and data replication, Replica Location Service (RLS) [25]. 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. A more detailed description of the GAP interface together with short example code can be found in [5].

3.3. Dynamically distributing Triana workflows

Triana provides a standard mechanism for distributing a group of tasks across multiple machines either in parallel or in a pipeline. Each group 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 runtime once they have information about the distribution policy and the number of services available.

Figure 1 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

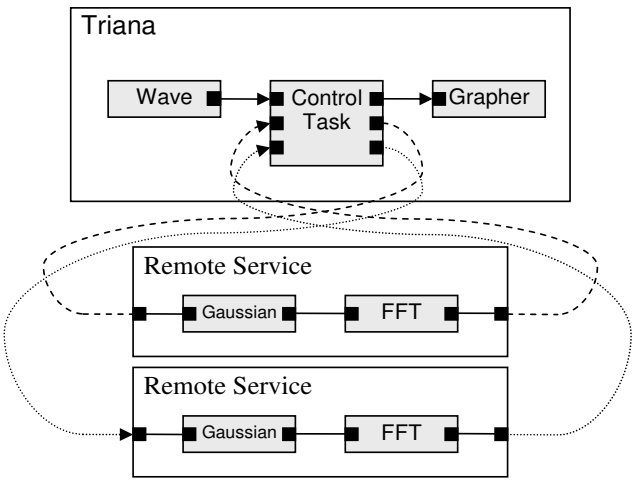


Figure 1. Triana dynamically rewires its workflow to connect to the remote services it has discovered through the GAP.

available services with data, passing new data to services that have completed previous analysis. The whole operation is asynchronous and the worker nodes (or distributed services) are free to work at their own speed. Users can specify how the data should be output from the control unit, either fixed in the order of arrival or, as and when data are received from the worker nodes. Depending on their choice, the control unit either buffers the data internally until the correct order can be resumed or passes the data directly out when they are received.

4. SEARCHING FOR COALESCING BINARIES USING TRIANA

This section explains how we have implemented a real-world scientific Triana example on the Grid. In this 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 running Triana services, and a combination of GAP Web service invocation and low-level library interfaces is used to connect to the GridLab data management system. We will first briefly describe the scientific goals of the scenario, followed by a description of how this was implemented in Triana and how this is mapped onto the testbed using the various services. Finally, we will explain how this example can be extended to support dynamic load balancing.

4.1. Inspiral search algorithm Triana implementation

One of the predictions of Einstein's theory of general relativity is the existence of gravitational waves. Gravitational wave detectors such as the laser interferometric detectors GEO600, LIGO and VIRGO

are attempting to search for evidence of these waves. One of the most promising potential sources is the radiation emitted by inspiralling massive compact binary systems. It is hoped that it will be possible to detect the radiation from the last few minutes before collision. However, the gravitational wave signal is very small, and complex data analysis techniques are necessary to extract the signal from the noise. Here we describe how Triana has been applied to search for waves generated by these systems.

To extract signals from the background noise, a technique known as *matched filtering* is applied. This involves generating thousands of templates (known as a *template bank*) which represent theoretical predictions of the expected signal, and fast correlating these with the signal. Each template contains different parameters defined at a certain granularity within the search space.

To perform this search in real time, it is necessary to have a computing resource capable of speeds in the range of 5–10 Gflops. The signal from the detector is sampled at 16 kHz and at 24 bits. However, the meaningful frequency range is up to 1 kHz; to achieve this upper (Nyquist) frequency, the data can be down-sampled to 2000 samples per second. The real-time data set is divided into sections of 15 min in duration, resulting in 7.2 MB of data being processed at a time, after re-sampling. These data are sent to a node, which generates its templates (a trivial computational step), and then processes the data. Typically, the template bank will contain between 5000 and 10 000 templates. This process would take up to 2 h on a 2 GHz PC running a C program; thus at least 20 PCs would be required to keep up with the data stream.

A rudimentary version of this search has been implemented in Triana. It loads 4 s of data, and performs a correlation with a generated inspiral template (also known as a *chirp*). The parameters for this template, the masses of the compact objects, are read in from an ASCII file, which acts as a simplified version of a template bank.

The correlation is performed by taking the Fourier transform of both the signal and the template, taking their product, dividing this by the power spectral density (PSD) of the noise, and taking the inverse Fourier transform. The correlation maximizes over the time of arrival of the chirp waveform. To maximize over phase, the data are independently correlated with two chirps, one of which is shifted in phase by 90° with respect to the other.

The implementation within Triana, explained in a recent publication [26], uses approximately 50 separate algorithmic components connected together to form a workflow (shown in the top right-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. This can be advantageous to application scientists in that it allows them to see exactly how the algorithm works in a simple, diagrammatic way, thus obviating the need to trawl through code to make sense of what is happening. The grouping mechanism allows the user to create group units out of simple units to perform more complex tasks. This allows the creation of re-usable algorithmic components. For example, in this algorithm, several individual units used to perform a correlation are grouped together to form a *Correlate* group unit. Another advantage is that new novel detection algorithms can be examined simply by inserting or replacing units in the workflow. This workflow is an example of the first of the types we defined in Section 1, serial scientific workflow for data processing.

4.2. Mapping the search onto the Gridlab testbed

Using Triana, the inspiral search workflow can be run on the GridLab testbed. This testbed consists of a large number of computational resources distributed across Europe. We have extended the Triana

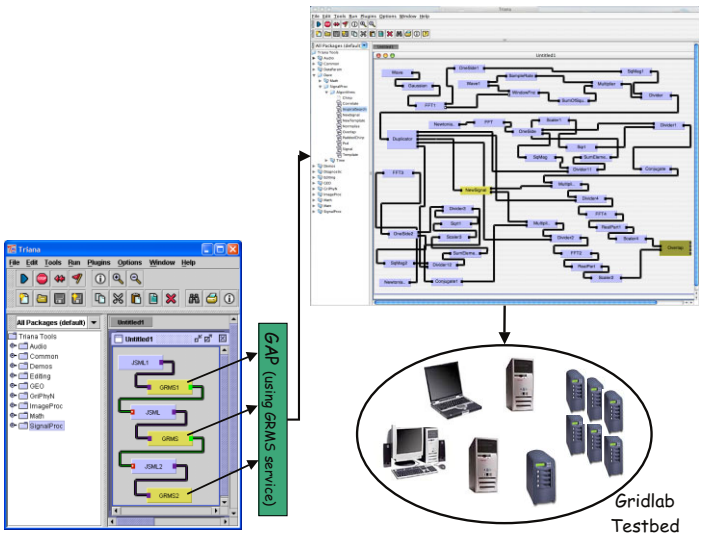


Figure 2. A workflow for the job submission of Triana services.

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 services that are deployed on the GridLab testbed. There are two services that we are currently interested in, the GridLab GRMS and the Data Management service. Both will be described in detail below.

4.2.1. Job submission: GridLab's GRMS

GRMS is a job submission service with a Web service interface deployed on the GridLab testbed. GRMS expects as input a definition of a job submission using the XML-based GridLab Job Definition (GJD) language. The GJD allows the user to specify the executable to be run and a number of optional parameters such as command line arguments, input and output redirection and URLs of where to find files needed for the execution of the job. GRMS is being extended to support the concept of workflow by allowing the user to chain simple jobs together into a single job submission. A Triana user may choose to use this capability, submitting a constructed workflow to GRMS for execution. Alternatively Triana can manage the workflow itself by dynamically submitting a sequence of individual jobs to GRMS. Leaving the workflow logic within Triana gives the user greater flexibility in workflow composition, enabling dynamic response to changes in the execution environment and the requirements of the workflow. For example, the user may decide to insert a local processing node into the workflow in response to data returned by a node monitoring the job execution, or even launch a new job on another remote machine. Furthermore, a GRMS workflow is not an extension to a GRMS simple job, and as a result, workflows cannot be nested. In Triana, on the other hand, groups of individual execution units

can be viewed as individual execution units in themselves, allowing for great flexibility in workflow composition. For example, a user may wish to define a taskgraph made up of individual job submissions and save that graph for reuse.

For these reasons each simple job submission to GRMS is represented in Triana as a single component. This can be connected to either subsequent job submissions that have a dependency on the execution of the job, to local processing nodes, or even other remote processes that are not invocations to the GRMS; for example, a Web service interface to a portal that can relay information on the status of the running jobs. Because all GRMS invocations return an *id* of the job just executed, we can feed this information into the next job, allowing us to chain independent simple jobs together on the Triana desktop. The GRMS Triana unit offers a graphical interface that allows the user to define the parameters of their job submission without having to deal with the details of writing GJD. This submission can optionally be saved to file as a GJD document, and read in again later. This is useful if the user has processes that are often repeated.

The GridLab services deployed on the testbed require user authentication using GSI-based X.509 certificates. We have extended our Web services implementation to allow for GSI communication which is virtually transparent to the user. Our Web service client will detect the GSI service automatically and attempt to locate a proxy object. If there is no proxy available or it has expired, the user will be prompted for their password. Providing the user has a valid certificate and key, the proxy will be generated and the invocation will be configured appropriately. Viewed from the level of the GAP, a GSI Web service is just another service which happens to require a password.

Implementing GSI in such a transparent way means we are able to use GridLab services from within a Triana workflow in the same way that we invoke ordinary Web services. Furthermore, the ability to chain independent job submissions together within the Triana GUI allows us to choreograph job submission workflow for complex submissions. In Figure 2 we show Triana using GRMS to start Triana services on the testbed that will then be used to run multiple copies of the inspiral search algorithm. The chain of job submissions involves a number of steps: CVS checkout, compilation, and instantiation of the Triana engine. Effectively Triana is submitting instances of itself to run on the Grid. This workflow is an example of the second type defined in Section 1, a job submission workflow that executes services on a testbed.

4.2.2. Data management

The detector data will be stored in a decentralized 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 data to be either locally or remotely loaded, depending on their location. In addition the RLS allows the use of logical filenames to retrieve the physical location of the data on the set of distributed resources. This provides geographical transparency by treating local and remote access identically. A logical file name is transmitted to a Triana service running the application, which uses the data management service to return the physical file location that is then used by the remote frame library to access the data.

Figure 3 shows the interaction between the desktop Triana 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

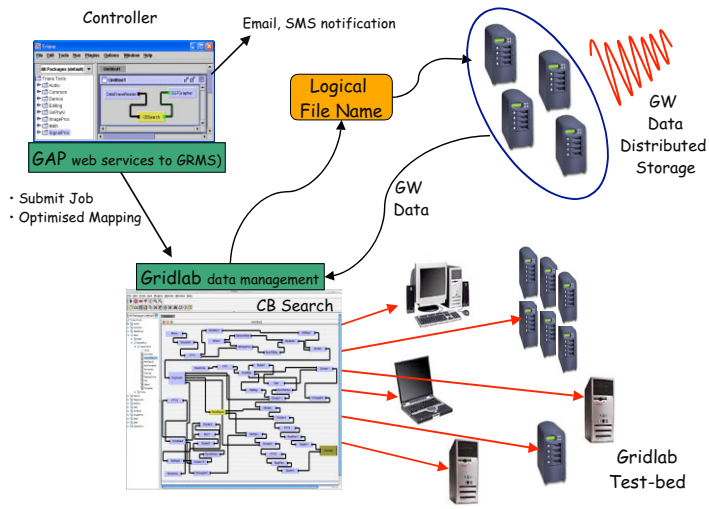


Figure 3. An illustration of the task-farming of the inspiral search algorithm and the interaction with the GAP and the Gridlab services.

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, 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 such as email notification, SMS notification and screen alerts, which are readily available as Triana components.

4.3. Dynamic resource allocation

An important aspect of the coalescing binaries search application is that it has real-time requirements. The gravitational wave detection device produces data in regular intervals, normally every 15 min. Since processing a single data block takes a significant amount of time, multiple compute resources are required to keep up with the detector.

In a Grid environment, the performance offered by the compute resources may vary significantly due to the wide range of processor speeds and architectures used. Machines may also simultaneously run multiple compute jobs. As a result, the performance of a single resource may vary over time, depending on the load that is generated by other (unrelated) processes.

An additional problem is that compute resources on the Grid can only be reserved for a limited time. After a reservation for a resource expires, a new reservation must be made. There is no guarantee, however, that this newly reserved resource offers the same performance as the resource it replaces. If its performance is lower, additional resources must be acquired to ensure that the real-time requirements

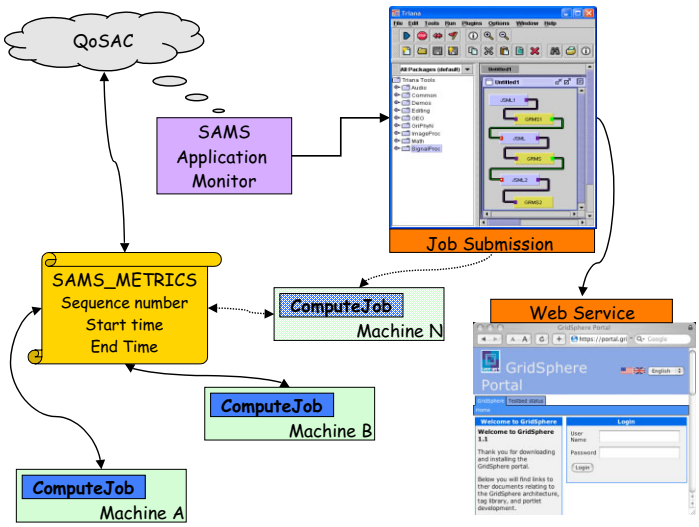


Figure 4. An overview of the simple application monitoring system.

of the application are met. If its performance is higher, however, other resources may be released to prevent wasting compute cycles.

To ensure that the real-time requirements of the application are met in the ever-changing Grid environment, a Simple Application Monitoring Subsystem (SAMS) is being developed within Triana to dynamically optimize the number of services that are employed. The SAMS itself is being implemented as a Triana workflow, since this environment is naturally suited to choreograph such decision-making applications. This is an example of the third and final workflow type defined in Section 1, a monitoring workflow that examines and modifies the behaviour of an executing application.

An overview of the current set of components used by SAMS is given in Figure 4. Briefly, each machine in the application repeatedly retrieves a block of data, processes it, and then generates an *application metric* to indicate that it has finished processing the block. These application metrics are simple data structures that contain performance-related information about the processing of the data block. They are propagated back to the SAMS using Mercury [27], the GridLab monitoring system. Using these application metrics, a special quality of service related adaptive component (QoSAC) inside the SAMS can decide if the current set of machines is meeting the requirements of the application, and reserve or release resources accordingly. We will describe the Mercury monitoring system and the QoSAC in more detail below.

4.3.1. Application monitoring

The Mercury monitoring system was developed as part of the GridLab project. It provides real-time application and resource monitoring facilities. Mercury features a multi-level consumer–producer

architecture as described in GGF's Grid Monitoring Architecture specification [28]. Mercury follows a modular design where all information providers (*sensors*) are implemented as modules that can be dynamically loaded at runtime, thus allowing easy configuration and extensibility.

Application management often requires that the manager component is able to influence the status of objects (e.g. services or running applications) that fail to meet some criteria. Mercury accomplishes this need by providing support for *actuators* that can perform operations on the monitored entities. Like sensors, actuators are also implemented as modules.

Mercury contains two elements to aid application monitoring and steering: an *application connector* module and an instrumentation library that communicates with this module. The instrumentation library provides an API for the application developer to register application-specific metrics and control operations. The registered metrics and controls are then made remotely available to Mercury consumers by the application connector module.

The application connector module is loaded by the Local Monitor daemon that is running on every computing node. Such Local Monitors are then grouped together by one or more Main Monitors usually running on a front-end node. This configuration ensures that the application does not need to know who is monitoring it, nor does the remote consumer have to care about on which local node the application is running. This also solves the problem that occurs when the nodes of a cluster have private network addresses or the cluster is behind a firewall, making direct communication between running applications and the outer world impossible.

When an instance of the inspiral search application is started within a Triana service, it registers a metric that generates an event after a data block has been processed. This registration is processed by the application connector module inside the Local Monitor. QoSAC can then subscribe to this metric at the Main Monitors of the Grid resources where the application is running. The Main Monitors forward the request to the appropriate Local Monitors and route back the events generated by the application to the QoSAC.

4.3.2. Adapting to quality of service constraints

Using the data provided by the monitoring system, the QoSAC is able to determine the performance of each of the individual resources. Using this information, it can then decide if the combined performance of the current set of resources is sufficient to meet the requirements of the application.

The current implementation of the QoSAC is able to adapt the resources used to meet a limited set quality of requirements. Currently, two types of task-farming scenarios are supported.

Deterministic: in this category of task-farming applications the number of operations required to execute the algorithm on the processing nodes is known in advance, it is fixed or can be computed from the input data. In this case, the user can simply specify the average time one chunk of data should take to process in order for the run to complete within a given time. The user will either have a set of data chunks or will be processing a continuous stream of data and will need for the computational nodes to keep up with the data rates. The coalescing binaries search application is an example of such a scenario.

Non-deterministic: in this category of task farming the algorithm that computes the data is non-deterministic. An example of this category would be artificial neural networks or parameter searches where an algorithm is traversing a multi-dimensional parameter space

and the completion of the algorithm is highly dependent on the space it is working within. Such algorithms are nonlinear and it is almost impossible to predict the number of operations required beforehand. A user cannot predict how many operations the algorithm will take to process the given data set, but may be able to specify how many operations must be performed per second to achieve an acceptable performance.

The coalescing binaries search application is an example of a *deterministic* task-farming application. In this application, a fixed number of operations are required to process each block of data produced by the detector. As a result, the time required to process a single data block is a direct measure for the performance of a resource. Also, to express the desired total application performance, a target average computation time per data block can be specified. From this target average computation time, the required block throughput for the application can easily be computed.

To adapt the number of resources used, the QoSAC calculates the average block throughput per resource and determines the sum of these averages to obtain the total block throughput of the resources. It then compares this total throughput to the required throughput. If the total throughput is below the requirement, the QoSAC generates an event to indicate that extra resources must be reserved. This event is received by a job submission component in the SAMS, which uses GRMS to start a Triana service on an extra resource. As soon as the Triana service is started, the QoSAC is notified so it can start collecting application metrics from this extra resource. If the total throughput is above the requirement, the QoSAC computes the surplus throughput, and tries to find a resource with the highest average throughput that does not exceed this surplus[§]. It then generates an event to notify the job submission component in the SAMS that this resource should be released. As soon as the resource is released, the job submission component notifies the QoSAC to allow it to clean up its administration.

Processing a single block of data may take a significant amount of time. For example, a modern 2 GHz PC reserved exclusively for this application requires at least 2 h per block. As we have explained earlier, both processor speeds and machine load may vary significantly in a Grid environment. As a result, processing times far exceeding these 2 h may be expected. This poses a problem for the QoSAC. When a new resource is added, it has no information about the performance of this resource, and it may take a significant time before any information is available due to the processing time required for a data block. While the performance information on the resources is not complete, it is difficult for the QoSAC to make a good estimate of the total application performance and decide if the current number of resources is adequate. In an attempt to solve this problem, the QoSAC estimates the number of data blocks that are waiting to be processed. Using the target block throughput and the runtime of the application, it can estimate the number of blocks that have been produced by the detector. Subtracting the number of blocks that have already been processed, the number of application metrics it received, and the number of blocks currently being processed, the number of resources, results in the number of blocks waiting to be processed.

Although a small number of waiting data blocks is desirable to prevent resources from becoming idle, a growing number of waiting blocks indicates a shortage of resources. Using this approach, the QoSAC is able to request extra resources when necessary, even if it does not have complete performance information on the current set of resources.

[§]This could be optimized to find a *set* of resources instead of a single one, but this is not currently implemented.

5. CONCLUSION

In this paper we have demonstrated the different workflow aspects used within the Triana PSE to implement a real-world, compute and data intensive, scientific application across a computational Grid testbed. We examined three distinct types of workflow that are combined to provide the scientific algorithm, job submission to the Grid testbed, and monitoring and adaption of the resulting application. We utilize 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.

As Grid computing moves toward the service-oriented paradigm of Grid services and Web services, users of these systems will increasingly want a simple layer of control above the services they are using. Workflow provides that control, and workflow composition applications, such as Triana, provide an intuitive interface for constructing and interacting with that control. Separating the scientific workflow from the job submission, and monitoring workflows insulates the user from much of the complexities of a running Grid application, without sacrificing the capabilities that a modern Grid environment provides.

ACKNOWLEDGEMENTS

The authors would like to thank the members of the application group for their invaluable contributions to the demonstration discussed in this paper, the GW group: Prof. Bernard Schutz, Prof. B. Sathyaprakash, Dr R. Balasubramanian, Dr Stanislav Babak and Dr Thomas Cokelaer. We would also like to thank the other members of the Triana team: Dr Omer Rana, Dr Roger Philp, Diem Lam and Shalil Majithia. Last but not least, thank you to all the partners in the GridLab project for their continued input.

REFERENCES

1. Foster I, Kesselman C, Nick J, Tuecke S. The physiology of the Grid: An open Grid services architecture for distributed systems integration. *Technical Report*, Open Grid Service Infrastructure WG, Global Grid Forum, 2002.
2. Allen G *et al.* Enabling applications on the Grid: A GridLab overview. *International Journal of High Performance Computing Applications (Special Issue on Grid Computing: Infrastructure and Applications)* 2003; **17**(4):449–466.
3. Hawick K, Elmohamed S, Dongarra J, Fox G, Meister J, Wilson GV. High performance computing and communications glossary. http://www.npac.syr.edu/nse/hpccgloss/ [July 2005].
4. GEO 600, June 2004. http://www.geo600.uni-hannover.de/ [July 2005].
5. Taylor I, Shields M, Wang I, Rana O. Triana applications within Grid computing and peer to peer environments. *Journal of Grid Computing* 2003; **1**(2):199–217.
6. Fox G, Gannon D, Thomas M. A summary of Grid computing environments. *Concurrency and Computation: Practice and Experience* 2003; **14**(13–15):1035–1044.
7. Taylor I, Shields M, Wang I. Resource management for the Triana peer-to-peer services. *Grid Resource Management*, Nabrzyski J, Schopf JM, Węglarz J (eds.). Kluwer Academic: Dordrecht, 2004; 451–462.
8. Taylor I, Shields M, Wang I, Philp R. Grid enabling applications using Triana. *Workshop on Grid Applications and Programming Tools*, held in Conjunction with *GGF8*, 2003. Available at: http://www.cs.vu.nl/ggf/apps-rg/meetings/ggf8/ggf8-ws-apps-upd.pdf.
9. W3C. Web Services Description Language (WSDL) 1.1. *Technical Report*, W3C, 2001.
10. SCIRun: A scientific computing problem solving environment. Scientific Computing and Imaging Institute (SCI), 2002. http://software.sci.utah.edu/scirun.html [July 2005].
11. Furmento N, Lee W, Meyer A, Newhouse S, Darlington J. ICENI: An open Grid service architecture implemented with Jini. *SuperComputing '02. Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society Press: Los Alamitos, CA, 2002.

12. Oinn T, Addis M, Ferris J, Marvin D, Greenwood M, Carver T, Wipat A, Li P. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics Journal* 2004; **20**(17):3045–3054.
13. Altintas I, Berkley C, Jaeger E, Jones M, Ludäscher B, Mock S. Kepler: An extensible system for design and execution of scientific workflows. *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM)*, 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004; 423.
14. Hoheisel A, Der U. An XML-based framework for loosely coupled applications on grid environments. *Proceedings of the International Conference on Computational Science 2003 (ICCS 2003)* (*Lecture Notes in Computer Science*, vol. 2657). Springer: Berlin, 2003; 245–254.
15. Krishnan S, Gannon D. XCAT3: A framework for CCA components as OGSA services. *Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004)*. IEEE Computer Society Press: Los Alamitos, CA, 2004; 90–97.
16. Slominski A, von Laszewski G. Scientific workflows survey. http://www.extreme.indiana.edu/swf-survey/ [July 2005].
17. Andrews T *et al.* Business Process Execution Language for Web Services Version 1.1.
18. Leyman F. Web Services Flow Language (WSFL) 1.1. *Technical Report*, IBM Software Group, 2001.
19. Condor Team. DAGMan: A directed acyclic graph manager. http://www.cs.wisc.edu/condor/dagman/ [July 2005].
20. Foster I, Kesselman C (eds.). *The Grid: Blueprint for a New Computer Infrastructure*. Morgan Kaufmann: San Francisco, CA, 1999.
21. Project JXTA, 2005. http://www.jxta.org [July 2005].
22. UDDI Technical White Paper. *Technical Report*, OASIS UDDI, September 2000.
23. Apache Project. Apache Web Services Project—Axis. http://ws.apache.org/axis/ [July 2005].
24. Czajkowski K, Foster I, Karonis N, Kesselman C, Martin S, Smith W, Tuecke S. A resource management architecture for metacomputing systems. *Proceedings of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*. IEEE Computer Society Press: Los Alamitos, CA, 1998; 62–82.
25. Chervenak A *et al.* Giggle: A framework for constructing scalable replica location services. *Supercomputing '02. Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society Press: Los Alamitos, CA, 2002.
26. Churches D, Sathyaprakash BS, Shields M, Wang I, Taylor I. A parallel implementation of the inspiral search algorithm using Triana. *Proceedings of U.K. e-Science All Hands Meeting*, Cox SJ (ed.). EPSRC, September 2003; 869–872. Available at: http://www.nesc.ac.uk/events/ahm2003/AHMCD/.
27. Gombás G, Balaton Z. A flexible multi-level Grid monitoring architecture. *European Across Grids Conference* (*Lecture Notes in Computer Science*, vol. 2970), Fernández Rivera F, Bubak M, Gómez Tato A, Doallo R (eds.). Springer: Berlin, 2004; 214–221.
28. Tierney B, Aydt R, Gunter D, Smith W, Taylor V, Wolski R, Swany M. A Grid monitoring architecture. *Technical Report GWD-PERF-16-2*, GGF, January 2002.

