Triana Generations

Dr. Ian Taylor
School of Computer Science, Cardiff University, UK
and The Center for Computation and Technology, Louisiana State University, USA
I.J.Taylor@cs.cardiff.ac.uk

Abstract

This paper discusses the Triana workflow system within the context of the workflow community at large. It provides a brief background for Triana and discusses the ways in which it has been used in the past for serial and as-well-as distributed tasks. A brief overview of other environments is given followed by a description of the Triana distributed architecture and a discussion of its key features, being: its user interface and its ability to work simultaneously in heterogeneous distributed environments. The high-level Grid and service-based interfaces that enable this support are outlined along with their corresponding bindings to the underlying middleware, such as WSPeer, Jxta, P2PS, Globus and Web and WS-RF services. New directions are given for Triana within the peer-to-peer context, followed by a description of two current uses for this technology and two collaborations, which are providing distributed P2P simulations to help test our P2P overlays for massively distributed processing and searching before deployment.

1. Introduction

Over the past five years, there has been a wealth of activity within the workflow domain for e-Science applications. On the one hand, there has been much activity from specific application domains, such as the support for scientists to conduct in silico experiments in biology through the construction of the Taverna workflow system [27]. On the other hand, there has been activity in more generalised systems, such as Kepler [26] or the VDS [10], for executing applications on the Grid in a generic manner. Triana has taken a different focus by trying to bridge the gap between different distributed environments for allowing true heterogeneous computing across different Grids and distributed paradigms. Such an approach has led to a number of different bindings to underlying middleware and therefore a number of possible modes of operation. A major design constraint for Triana has been to make it easy to extend into other distributed environments and therefore, its distributed specifications and functionality are subject to common cross-environment interfaces that are capable of switching between environments at run time. These interfaces are discussed in section 3.1. Triana can be used in a fine-grained way, specifying dataflow for local operations and in a course-grained fashion for distributed workflow (see Section 2.1).

A key strength of Triana is its graphical user interface. It has evolved in its Java form for over 10 years and was built using lessons learned from its previous C++ counterpart that was developed several years before. Triana contains extremely convenient and powerful editing capabilities, both for specifying workflows and units, and contains a number of wizards for on-the-fly creation of tools and their user interfaces.

In this paper, the core Triana components are discussed and new perspectives within the peer-to-peer (P2P) domain are outlined. Existing and new Triana applications are given along with empirical measurements from distributed simulations that are helping to verify its distributed behaviour before deployment. In the next section, Triana is introduced and some of its projects are listed along with a discussion of the flows it can represent and how it may be be used for project development. In Section 3 an overview of Triana’s distributed functionality is given. Finally, in Section 4, new directions and applications are presented, along with current simulation activities.

2. Triana

Triana$^1$ is not a monolithic system targeted at a specific user base or application domain; rather, it is a collection of components which is designed be applicable to a number of different application scenarios and environments. In terms of flexibility, Triana is arguably one of the most sophisticated tools available. Triana can be many things in different situations. It is a problem-solving environment, being capa-

$^1$http://www.trianacode.org/
ble of acting as a complete, integrated computing environment for composing, compiling, and running applications in a specific area [18]. An example of this is given in the following subsection. It can also be used as a dataflow system, a distributed-workflow or workflow-management system, or an automated scripting tool. Triana has been or is being used in the following domains: gravitational wave analysis [9] (also see below), radio astronomy\(^2\), astrophysical simulations [21], data mining [1] [5], biodiversity problems [4], galaxy visualization [34], audio processing and distributed music information retrieval (MIR) [29], distributed peer-to-peer simulations [30], grid-enabled medical simulations [2], environmental science [3], fleet management [22] and e-health [23]. It can be used as a graphical interface for the composition of Web Services [33] and can switch between Grid and P2P environments [33]. It has also spawned several new projects, focusing on distributed middleware for accessing P2P and Web services [24] [35].

2.1. Triana is Workflow or Dataflow?

Triana was designed in 1990 to be a quicklook data analysis system [32] for the GEO 600 gravitational wave project [19]. The original system was specified by Bernard F. Schutz (one of GEO 600’s principal investigators) along with other scientists from the project with the aim of performing on-site on-the-fly analysis of gravitational wave data being collected from the GEO 600 laser interferometer located near Hannover, Germany. Data from the detector is collected through sampling, much the same as audio data, but at much lower sampling rate. The result is that there are hundreds of signals sampled at varying rates, representing science and housekeeping data. The scientist would like to view and perhaps perform some preliminary data analysis, typically for detector characterisation for locating potential problems with the data or identifying certain events. The types of tools required for this include the standard mathematical tools and tools for statistical analysis, transformations (FFTs), filters (e.g., low, high and band pass) and for the correlation of data sets. Triana therefore contains a number of tools (around 400) for the analysis and manipulation of such data, which are all written in Java (with some in C). Ironically, the original system was called Grid [31] for the first several years until the word became popularised by another field . . .

Tools, which are called units in Triana, are represented by boxes on a work-surface and are connected by cables. These cables actually represent the type of data that is flowing between the units as each unit specifies which data types it can process. When units are connected a user drags a cable from the output of the sending unit to the input of the receiving unit. The underlying type-checking system checks that the data types acceptable by the receiving unit are compatible with those from the sending unit. If so, the cable is drawn indicating that a connection has taken place, if not, then an error message is displayed by listing the data types to aid in debugging. A screen-shot illustrating a visual representation of such a pipeline is given in Figure 1.

In this context, Triana is a dataflow system. The dependencies between the units are specified through data interactions and therefore the synchronisation of the flow is data dependent. However, this is only one use of Triana, which focuses on the serial execution of fine-grained algorithms for the application of a specific task. Other Triana uses, for example, for choreographing Web or WS-RF services, focuses on the flow of work between course grained distributed services. Within this context, I would argue, Triana is used a workflow system rather than a dataflow system because the dependencies between the invocations are for controlling the flow of the execution of such services rather than specifying their data interdependencies directly. The flow of work involves executing Web services and the movement of data is generally achieved through a behind-the-scene data-management system. In such course-grained scenarios, I do not believe therefore the term dataflow is appropriate because rather sensibly one would not want to pass actual data between the services (the overhead is too high); rather, the services would use a separate optimised data-management system to pre-stage or transfer the data to the appropriate

\(^2\)http://www.gridoned.org/
distributed resource before execution. Therefore, the actual movement of data is simply another control flow within the set of operations you wish to perform. Of course there is an implied data dependency but because this is handled by a data management system, the flow of work is actually between services and data management systems rather than the services processing the data directly. In most distributed Web services or Grid scenarios the management of data is de-coupled from the actual services and therefore it no longer represents a dependency in the context of the entities within the flow of work to be performed. The flow of work is therefore focused far more on the execution of the services which are dependent on other services, rather than on the the execution of services that are dependent on data.

Within this context therefore, Triana can be used as a dataflow or distributed workflow system, depending on the environment and application it is used for. Other systems focus more on distributed workflow directly [27] [10] [12] [17] [11].

2.2. Why use Triana?

Triana was originally designed to help the gravitational-wave scientists to be able to spontaneously create and run data analysis algorithms on the data at its source. There are several reasons why a system like Triana was thought to be desirable. Such a component-based object-oriented approach allows scientists to easily create new algorithms that conform to an agreed and defined set of data types and can adapt to different internal parameters e.g. size, resolutions etc. This coupled with the graphical interface means units can be connected arbitrarily without the need for re-compilation, which would be the case if non-conforming algorithms were used. Therefore algorithms only need to be integrated once. By designing components in such a way, it makes it easy to create individual user interfaces for each component to allow the modification of its internal parameters. This is highly desirable for quick-look dataflow assembly and Triana has a built-in Wizard to help programmers build user interfaces without writing the code directly. Such an approach also allows users to organise units/components graphically within toolboxes in a directory-type structure for easy browsing. These features coupled with the graphical approach allows simple type-safe orchestration of data analysis pipelines on-the-fly without the need to manually reconfigure the code and also encourages reusability of existing units.

Such elementary design features transfer across when used with distributed components, such as Web services. In the service oriented architecture (SOA) sense, Triana facilitates the publish, find, bind triad of operations that enable impromptu assembly of distributed software assets. Further, through its editing capabilities, discovered services can be renamed, organised in directories and grouped to allow users to customise their environment. The distributed capabilities coupled with its existing units allows the creation of highly complex workflows that contain local Java units connected to distributed services, which can produce high-quality presentations of results through the use of one of the several graphical output units available. There are in total around 600 local units available for use within the core distributed of Triana, which span many domains, such as signal, image text and audio processing, and a number of internal scripts for dynamic reconfiguration of distributed workflows. Throughout the years of development, Triana has accumulated a number of features and has become graphically extremely powerful. Much care has been taken in the design of the usability of its GUI for connecting, editing (unit parameters and workspace drag and drop, copy/paste, etc) and for grouping (aggregating) components on-the-fly without the need for recompilation. Triana is highly configurable and customisable and can be used graphically or on the command line.

Combining its features, Triana is much more than a workflow or dataflow system. Rather than being aimed at an individual level Triana is more aimed at the project level. It encompasses many tools than enable project scientists to design and create systems than connect together a number of software components, whether they are local, legacy applications, distributed P2P, Web or WS-RF services, or whether they wish to integrate their project with Grid computing capabilities, such as GRAM, GridFTP, etc. New projects can be constructed on-the-fly and shared with the consortium as a whole. In this sense, Triana is not only a workflow management system but a project management system also.

3. Triana for e-Science Workflows

Triana has the capability of running a number of different software components and distributed services on the same work surface. The following list contains the types of components and services Triana can interact with, which is followed by a list of the technologies Triana integrates in order to achieve this functionality:

Java Tools: Triana can instantiate any Java component, which can be connected with any other if they extend certain interfaces for specifying input/output types and have access to the functionality for data passing.

Legacy Applications: Legacy applications can be wrapped locally or in a distributed setting and can be monitored dynamically by using the GMS framework.

Grid Tools: Triana has a full binding to represent the Java GAT interface visually within a workflow, which is ca-
pable of invoking tools and services such as Condor, GridFTP, GRAM, etc.

**P2P Services:** Through the GAP Jxta3 and P2PS bindings, P2P components can be discovered and invoked using the framework.

**Web Services:** The GAP binding to WSpeer allows Triana to interact with Web Services and a number of other Web service specifications, such as WS-Security, WS-Addressing and WS-Transfer and WS-RF (including WS-Notification).

**Distributed Group Tool Deployment:** Any connected combination of the above tools and services can be grouped and deployed on a distributed resources, using the GAP deployment mechanism, as a Web WS-RF or P2PS service.

### 3.1 The GAT and GAP High Level Interfaces

Triana interfaces with the Grid Application Toolkit (GAT) [6] and its peer-to-peer subset, called the GAP [33], in order to integrate with both Grid tools and distributed services, respectively. The GAT exposes application-level capabilities along with adapters that adapt these capabilities to the low level mechanisms, provided by the middleware e.g. Globus [20], Condor [16] etc. The GAP is a simple publish, find, bind interface similar in spirit to SOA concepts, for advertising, discovering and communicating with resources within service-oriented networks. An overview of this architecture is given in Figure 2. Here, we illustrate that the GAT is used for Grid computing services, whereas service-based communication is accomplished by the GAP and its bindings. The GAP is also used to provide the fine-grained distribution mechanisms for running Java units remotely as services or as collections of services.

### 3.2 The GAT and GAP Bindings and Tools

The GAT interface is represented visually in Triana. A user can drag and drop Grid components, and specify a workflow of dependencies between Grid tools for running jobs (e.g. GRAM, GRMS, Condor etc) with units that represent files that need to be staged (e.g. using GridFTP) before execution of the next task. Such an implementation states explicitly the concept of workflow argued in Section 2.1 because any dependencies are work oriented, not data oriented directly. Here the file units represent data dependencies implicitly but they explicitly represent work to be performed, e.g. move a file to a distributed location. An example workflow is show in Figure 3. Here, the user effectively interacts using the standard Triana unit (component) and connection (cable) paradigm, where any unit (or group unit) can be submitted as a Job and the cables represent file movements to and from resources.

A tool that is often used in conjunction with the GAT workflow is gridMonSteer (GMS) [21]. GMS is a Java wrapper that is used to run an application and monitor its file output. Typically, GMS is submitted as a job using GRAM or similar and takes the application details as a program argument, along with information about distributed listeners for events. When GRAM executes GMS, GMS in turn, runs the application and starts monitoring its output directories. When the application writes a file, GMS detects it as an event and notifies any distributed listeners. If the distributed listeners are interested in this file, then GMS passes the file to them via a Web service interface, which is dynamically hosted by WSpeer, described below. Therefore, GMS can be used to monitor applications and it is extremely useful for iterative computations where intermediate results are

![Figure 2. An overview of the distributed architecture of Triana and its interaction with the GAT and GAP interfaces.](image)

![Figure 3. An Example Visual GAT workflow consisting of local Visual components, file dependencies and job submission.](image)
typically written as files. In the scenario described in [21], we use GMS to pass 3-D images from a distributed black hole simulation for visualisation within Triana.

The GAP on the other hand has bindings to three service-based middleware: P2PS, WSPeer and Jxta. The Peer to Peer Simplified (P2PS) [35] binding is a lightweight P2P system, which was written in response to the complexity and overhead associated with Jxta. P2PS implements a simple yet generic API for developing peer-to-peer applications. P2PS encompasses intelligent discovery mechanisms, pipe-based communication and makes it possible to easily create desirable network topologies for searching, such as decentralised ad-hoc networks with super peers or rendezvous nodes. P2PS is designed to work within dynamic environments, where peers maybe highly transient.

P2PS was motivated by Triana applications and then further driven by the needs of the NRL SRSS group (see Section 4.3). Triana needed a reliable P2P middleware for some internal projects and the SRSS were investigating the simulation of P2P discovery mechanisms within MANET networks but found Jxta to be too complicated a system to hook into and replace the transport layer. Therefore, P2PS was initiated as an extremely simple but effective P2P middleware for the GAP and thereafter it was expanded through the introduction of new Triana applications and SRSS discovery and grouping requirements. Although, P2PS contains less protocols than Jxta, the key components for operating in self-organising dynamic networks are integrated. The P2PS reference implementation is written in Java and is a fraction of the size of the Jxta codebase but it includes multiple endpoint resolvers (e.g. TCP, UDP, Multicast, SSL, etc), virtual pipes, relays that bridge between protocols (e.g. to traverse a firewall), XML-based messaging. Rendezvous nodes for creating super-peer networks and groups, which implement security and provide a scoping environment for its peers.

The WSPeer GAP binding [25] allows the creation, publishing, discovery, invocation and composition of Web and WS-RF services within Triana workflows. WSpeer itself is a high-level interface to core Web services abstractions but it has been designed specifically with a P2P context in mind. Therefore, WSpeer can host Web services as peers; that is, service providers and consumers, and allows an application to add itself as a listener to events fired when Web services are published, created, or invoked, rather than relying on a container model. WSpeer supports the conventional HTTP-based binding and UDDI, but also supports Jxta, P2PS, and the Styx protocol\(^4\). The bindings to P2PS and Jxta in WSpeer allows Web and WS-RF services to be hosted within a P2P environment, where decentralized publishing and discovery can be used. WSpeer can also use logical addressing to identify endpoints and therefore work in environment that operate outside the scope of IP addressing and DNS. This is important for our simulation binding of P2PS, described in Section 4.3, which use NS-2 addressing (integer identifiers) to address the nodes. On top of the HTTP and P2P environments, WSpeer supports Web services and WS specifications including WS-Security, WS-Addressing and WS-Transfer and WS-RF (including WS-Notification).

### 4 New Directions for Triana

Since 2001, the Triana project has been working with P2P technologies. We developed the P2PS middleware, just discussed, and then we built WSPeer to bridge Web services and P2P technologies to allow the hosting of Web and WS-RF services within ad-hoc P2P networks. We are currently in the process on building on these technologies to create a standards-based Web services P2P overlay (code-named the Alchemist), which is built on WSPeer abstractions over P2PS [25] that will be capable of hosting a number of future-looking Triana applications. The Alchemist is basically a toolkit for creating decentralised overlays that may contain super peers (for caching adverts) or other types of peers (such as data centers mentioned in Section 4.3 for caching data across BOINC-type networks). The Alchemist also provides monitoring tools and a workflow toolkit based around Triana that allows workflows to be dynamically deployed across the network for maintainability and upgrades. The framework is being built in order to support a number of applications that we are currently working on within the group. In this section, I discuss the background into P2P and Grid computing and some applications that have motivated the design of the Alchemist. I then discuss some preliminary simulation results to indicate its performance predictions and a generic simulation environment, which will be used to simulate the Alchemist applications before deployment.

#### 4.1 The convergence of Grid and P2P computing?

Over the past five years, there has been significant advancements made in the field of distributed systems technologies with applications being deployed on an Internet scale. On the one hand, Grid computing has evolved from core toolkits, such as Globus\(^5\) providing the underlying mechanisms for executing and managing Jobs and data, to using service oriented architectures (SOAs) in the form of Open Grid Services Architecture (OGSA) [15] that capitalises on Web Services standards [8] for exposing the low level functionality on a network. The recent advancement to WS-RF [13] provides decoupled mechanisms for represent-

---

\(^4\)http://sourceforge.net/projects/jstyx

\(^5\)http://www.globus.org/
ing stateful resource capabilities through stateless Web Services interfaces, which allows a system to manage lifetime without using a tightly coupled approach like previous distributed object systems, such as Corba\(^6\) or Jini\(^7\). Grid computing is built on standardised technologies and extended through other standardisation efforts often initiated through the various research and working groups hosted within the Global Grid Forum (GGF) conference \(^8\).

Peer-to-peer (P2P) technologies, on the other hand have grown through grass roots Internet culture initiated by specific applications, such as file sharing systems like Napster\(^9\) or Gnutella\(^10\) and CPU sharing systems like SETI (Search for Extra Terrestrial Intelligence\(^11\)). At its essence, P2P is about connecting people so that users can share information or participate in projects by being offered a various range of incentives to do so. P2P therefore is much more focused on solving scalability and robustness issues when faced with huge numbers of connected participants, which are extremely unreliable in that they can disconnect frequently and are hosted behind application-unfriendly mechanisms, such as Network Address Translation (NAT) systems and Firewalls. With such a high number of highly transient participants, writing applications for such a networking environment requires three key design issues to be tackled in order to cope with such a dynamic environment: scalability, reliability and interoperability.

The growth of these two largely independent areas has been exponential with enormous momentum and diversity. However, there is a necessary convergence that is taking place as production Grids move forward into deployment over significantly more participants. The scalability techniques offered through P2P algorithms that not only balance the load, through encouraging the use of servants, where each peer is both a client and a server, but also address robustness through the use of more decentralised network overlays that can adapt to random failures across the network. Such observation have been made in several papers, e.g. [7] and [14].

4.2 Applications

One current project that will use the Alchemist framework is DART (Distributed Audio Retrieval using Triana), which is being researched jointly between Cardiff University and the Laboratory for Creative Arts and Technologies (LCAT) in Louisiana State University. DART capitalises on the Triana developments to provide a decentralised overlay for the processing of audio information for application in Music Information Retrieval (MIR). The system will make use of an Alchemist overlay under development that provides a P2P overlay similar to the BOINC\(^12\) paradigm. Under this general scenario, users provide CPU cycles to projects to help in the analysis of data for searching of scientific or non-scientific events. Within the DART scenario, users provide metadata to the network that is created through the analysis of the audio files located on their individual hard drives.

The local processing on the network participants will be achieved through the use of Triana, which allows the coarse-grained specification of data flows, to provide pipelines for processing the data in specific ways. Triana would enable extremely rapid prototyping and code reusability of analysis algorithms exposed by a graphical interface, where units (representing audio analysis tools) can be dragged, dropped and connected together in order to create the algorithms. Such algorithms can be deployed i.e. uploaded to the peers in the network, through the use of decentralised P2P structures that would enable caching and file replication for speed and efficiency.

Another project “Secure Decentralised Data Sharing in Dynamic Distributed Networks” (EPSRC EP/C006291/1) is focusing on providing decentralised data access to CPU sharing applications. The work builds on the Alchemist framework in order to address data scalability issues within large scale distributed computing platforms by using a P2P approach to allow the system to scale proportionately as the load or number of users increases, thus making the system completely dynamic and self-organising. The project plans to integrate its findings into the BOINC system and is driven in part by requirements from the Einstein@home BOINC project. The key advantage of such a data distribution approach is that maintenance and administrative overhead is virtually non-existent and therefore it becomes extremely inexpensive to run. Some preliminary simulations already conducted with ICAR-CNR, discussed in the next section, indicate that indeed such an approach is highly scalable.

4.3 Distributed Simulations

The Triana project is involved in collaborations with two independent groups working on simulations of P2P networks. One project is in collaboration with the Naval Research Group in Washington DC, USA and the other with ICAR-CNR, Italy.

With NRL, we have been developing AgentJ \([30]\), which provides the unique ability to simulate real-world Java networked applications within the NS-2 Network Simulator\(^13\). It supports transport protocols (e.g. UDP unicast and multicast) and timer utilities, which are commonly used in real-
world network applications and simulations. AgentJ extends the NS-2 network simulator through run-time bytecode rewriting to simulate unmodified Java network applications. For application developers, AgentJ provides a distributed simulation environment for Java applications for performance analysis and optimization.

One of the main focuses for AgentJ has been to facilitate the simulation of Java peer-to-peer (P2P) architectures, such as P2PS [35] and Jxta. To this end, the main driver for the project has been the P2PS middleware, which is also used as the underlying infrastructure for the Alchemist project. P2PS is now running without modification in AgentJ and we have run a number of preliminary simulations, which will be submitted for publication soon.

With ICAR-CNR, we have been working on the simulation of the decentralisation of data within BOINC-style networks. In this scheme, we identify a Job initiator that sends the data once to the network, then the network propagates this across the data nodes as and when required (by distributed caching). In BOINC-like networks, there is often the need to send a data file to several workers due to the unreliability of the nodes. This replication can be sped up through the use of such a data caching mechanism. Furthermore, some projects require many nodes process the same data by performing parameter sweeps. Data delivery can be expedited significantly in this case by using such an overlay.

A set of simulation runs were performed to evaluate the impact of the caching and replication mechanism on a set of performance indices, such as overall time to execute the jobs, throughput, mean time to download a data file, and load experienced by data centers and worker nodes. Results have shown [28] that indeed such a distributed caching mechanism significantly improves performance.

4.4. Conclusions

In this paper, I discussed the motivation in the use of the Triana environment for problem solving and orchestrating flows of operations or services. I discussed its role both in fine-grained dataflow applications and as a coarse-grained distributed workflow system, and provided many current examples by listing projects currently using Triana. By expanding on these ideas, I then described the underlying components that comprise Triana as a whole by discussing the various tools and services that can be used in a number of different distributed environments, and which are accessed through the high-level application interfaces, the GAT and GAP.

I then outlined some new directions for the Triana project by focusing on its current peer-to-peer capabilities and how these are being exploited to build a framework, called the Alchemist, for hosting a number of different ubiquitous distributed applications. I provided the motivation for such developments by discussing the evident convergence between peer to peer and Grid computing. Finally, I outlined two such applications that are being tackled using the Alchemist in distributed CPU scavenging and distributed music information retrieval.

4.5. Acknowledgements

I’d like to thanks our sponsors, PPARC, EU and EP-SRC for supporting this effort through the GridOneD, Grid-Lab and EP/C006291/1 project, respectively. I’d also like to thank the Post Doctoral programmers (Matthew Shields, Ian Wang and Tom Goodale) and Ph.D students (Andrew Harrison, Ian Kelley and Eddie Al-Shakarchi) at Cardiff for realising these components and our collaborators in ICAR-CNR, NRL and CCT for being involved in the joint projects described in this paper.

References


GEO 600 aims at the direct detection of gravitational waves. http://www.geo600.uni-hannover.de/.


