Triana as a Graphical Web Services Composition Toolkit

Shalil Majithia, Ian Taylor, Matthew Shields, Ian Wang
Cardiff School of Computer Science, Cardiff University, Cardiff
{shalil.majithia, i.j.taylor, m.s.shields, i.n.wang}@cs.cardiff.ac.uk

In this paper, we extend the functionality of the Triana problem-solving environment into the Web services world. Triana uses a peer to peer subset of the GridLab GAT interface, that we have labeled the GAP Interface. The GAT/GAP are middleware independent APIs that allows transparent access to various underlying middleware architectures. Current GAP bindings include JXTA and P2PS, an in house lightweight P2P toolkit. Here, we newly extend this functionality to create a Web services binding for the GAP, which allows users to graphically and transparently create Web services workflows. In particular, we look at the discovery, invocation, composition, and publishing of Web services using the Triana environment. Such composed applications may then be written as BPEL4WS graphs by using the Triana pluggable architecture and executed from within Triana or any Web services choreography engine.

1.0 Introduction

Web services are emerging as an important paradigm for distributed computing [1,2]. It is argued that services will be composed as part of workflows to build complex applications to achieve client problem requirements [3,4]. The composition of services into workflows raises two challenges. First, the users have to invoke several services individually (maybe running into hundreds) and coordinate data transfer between these. This is a tedious task and requires the user to be aware of the low level details of accessing registries, invoking services, and troubleshooting in case of problems. Although it is possible to automate these tasks by writing scripts, the users still need to have an in-depth knowledge of writing, running, and troubleshooting scripts. Second, the whole process needs to be repeated afresh each time the composite process is run. This makes it difficult to validate or share the process.

From the above discussion, it is clear that it is an important challenge to make it possible for users to construct complex workflows graphically and transparently and thereby insulating them from the complexity of interacting with numerous heterogeneous services. Based on these requirements, a Web services composition system needs the following mechanisms:

- Service discovery methods: there must be a mechanism by which a user, and other components, can locate relevant services on the network;
- Service composition methods: there must be mechanisms to allow composition of services in a simple graphical manner;
- Transparent invocation methods: there must be mechanisms to invoke services transparently, i.e. without requiring the user to develop a client for each service;
- Transparent publishing of services: there must be mechanisms to allow users to publish composite services.

These mechanisms are the basic requirements of an architecture for building complex distributed Web services based systems. By facilitating the transparent construction of Web services workflows, users can:

- Create new composite services which offer more functionality than atomic services;
- Share and replicate workflows with other users;
- Easily carry out 'what-if' analysis by altering existing workflows;

In this paper, we propose a solution which aims to provide such an architecture by extending the open source Triana problem solving environment.

The remainder of the paper is organized as follows: Section 2 provides an overview of Triana and the GAP API. Section 3 describes the components of the WServe API and Section 4 outlines the current status and planned future work.

2.0 System Overview

This section presents an overview of the system explaining the Triana environment, how Triana uses the GAP interface, and finally how WServe implements the GAP binding for Web services.

2.1 Triana

Triana [5,6,7] is an open source, distributed, platform independent Problem Solving Environment (PSE) written in the Java programming language. A PSE is a complete, integrated computing environment for composing, compiling, and running applications in a specific area [8]. The Triana PSE is a graphical interactive environment that allows users to compose applications and specify their distributed behavior. A user creates a workflow by dragging the desired units onto the workspace and interconnects these by dragging a cable between them. Although, the focus here is on the graphical interface, Triana consists of a complex set of interacting components that create the complete system or any subset. This federated approach gives Triana the flexibility it needs to be able to be applied to many different scenarios and at many different levels. For example, it can be used as a workflow engine for grid applications; for connecting data driven grid components and managing the workflow between them, or as a data analysis system for image, signal or text processing applications; allowing a scientist to quickly apply algorithms to data sets and view results. It can also be used as a high-level graphical script editor for creating a number of task-graph or workflow script formats including, but not limited to Directed Acyclic Graphs (DAG) and BPEL4WS [9] formats (see Figure 1).

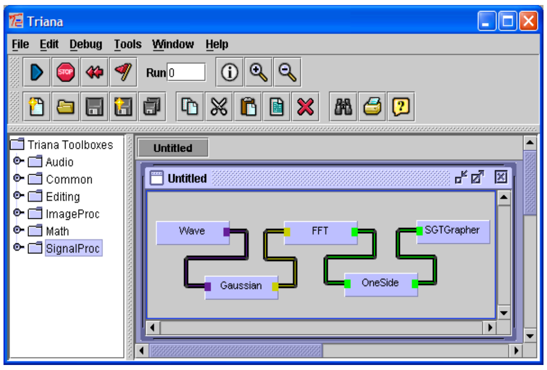

Figure 1: The Triana user interface representing a simple signal processing network.

Triana was originally developed for the GEO600 gravitational project [10]. Here, it was, and still is, used as a quick-look data analysis system for analyzing gravitational wave signals that are output from the laser interferometer detector located in Germany. During the past two years it has been restructured and redesigned in this new federated architecture. Briefly, the user interface has been completely disconnected from the underlying subsystem for both the functionality of the main system and for every Triana unit and its associated user interface. Clients (i.e. those running a GUI) can log into a remote Triana Controlling Service, build and run the Triana network remotely and then locally visualize the result on their device even though the visualization unit itself is run on the remote server. Users can also log off without stopping the execution of the network and then log in at a later stage to view the progress (perhaps using a different device e.g. mobile phone, handheld). In this context, Triana could be used as a visual environment for monitoring the workflow of Grid services or as an interactive portal by running the Triana Controlling Service as a Servlet on the Web server and by running the applet version of the Triana GUI. Further, Triana networks can be run as executables in a stand alone or batch processing mode since any Triana network can be run with or without using the Triana GUI.

2.2 The GAP Interface

The GAP Interface is a straightforward API for peer-to-peer networking, providing applications with methods for advertising, locating, and communicating with other peers/services.

The GAP Interface is, as its name suggests, only an interface, and therefore is not tied to any particular Grid middleware implementation. This provides the obvious benefit that applications written using the GAP Interface can be deployed on any Grid middleware for which there is a GAP binding, as shown in Figure 2. This also means that as new middleware, such as OGSA [12], becomes available, GAP-based applications can seamlessly be migrated to operate in the new environment.

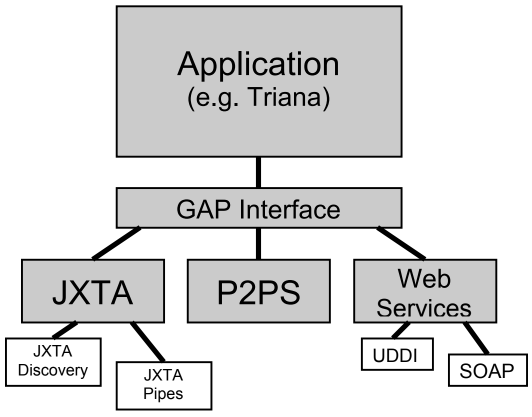

Figure 2: The relationship between Triana, the GAP Interface, and GAP bindings

Currently there are three middleware bindings implemented for the GAP Interface: JXTA, a peer-to-peer toolkit originally developed by Sun Microsystems [13]; P2PS, a locally developed lightweight alternative to JXTA; and a Web services binding, the implementation of which is discussed in this paper (see Section 3). As mentioned earlier, the GAP Interface provides methods for advertising, locating and communicating with other peers/services; however the mechanism used to provide this functionality depends on the individual bindings. For example, in the JXTA binding, service discovery is done through the JXTA Discovery Service, while in the Web Services binding, discovery is done using UDDI (as illustrated in Figure 2).

As well as providing sufficient functionality in itself to enable the construction of peer-to-peer applications, the GAP Interface was also designed as a prototype for the peer-to-peer subset of the GridLab GAT-API. GridLab is a pan-European project that is developing an easy-to-use, flexible, generic and modular Grid Application Toolkit (GAT) [11]; Triana is a test application for this project. When released, the GridLab GAT will provide a consistent API to resource brokering, monitoring and other services through an adapter architecture. It is intended to implement a GAP-GAT adapter enabling GAP and its bindings to provide peer-to-peer services to GridLab GAT applications.

2.3 WServe: GAP binding for Web services

WServe is an API that implements the GAP binding for Web services (Figure 3). Specifically, it provides the functionality needed to discover, invoke, and publish services. Services are discovered by querying UDDI based on user specified attributes. Services are invoked through a gateway that also handles data type conversions. Composite services can be published to the network by executing a simple GUI based wizard.

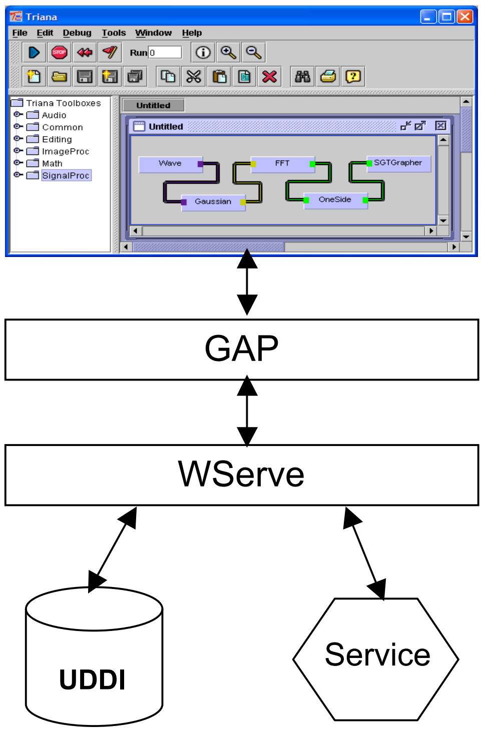

Figure 3: The relationship between GAP and WServe

3.0 The WServe API

As introduced above, WServe provides the key functionality to integrate Web services with Triana. In particular, WServe facilitates service discovery, invocation, and publishing.

3.1 Discovery of services

Triana services are discovered by querying a UDDI server. A Search can be done on user specified parameters and only those services matching the search parameters are retrieved. WServe provides classes that will search the registry, read the WSDL location, and retrieve the WSDL document. The WSDL document is then read and a Triana tool representing the service is instantiated. This tool is a proxy for the Web service. On one hand, it interacts with other tools in the composed graph and on the other hand it interacts with the WS Gateway to make the service calls. Interaction with other tools may be receiving or passing on data, or control instructions e.g. for looping. Finally, the toolbox is populated to indicate to the user that the service is available and can be used.

3.2 Composing Services

Services are composed simply by dragging and dropping the tools required from the toolbox onto the canvas and connecting them with pipes. This composite graph can then be executed or saved in a format for which a writer is available. We have currently implemented writers in a Triana proprietary format and BPEL4WS. Additionally, it is also possible to read in a workflow which may have been created by another user. Triana can handle any workflow as long as it is written in a format for which a reader is available. We have currently implemented a Triana proprietary format reader and a BPEL4WS reader which can be used to read and execute the graph.

3.3 Invoking services

Services are invoked from Triana through a WS Gateway. The Gateway uses a custom serializer (ObjectMarshaller) to convert proprietary Triana data types into Java primitive types or JavaBeans; this will eventually incorporated into the SOAP custom serilization mechanism. The Triana tool representing the service passes on the WSDL document to the Gateway. The Gateway dynamically configures the call, and invokes the service. The results are converted back to the Triana proprietary data types by the ObjectMarshaller and returned to the tool (see Figure 4). The invocation is done using SOAP over HTTP.

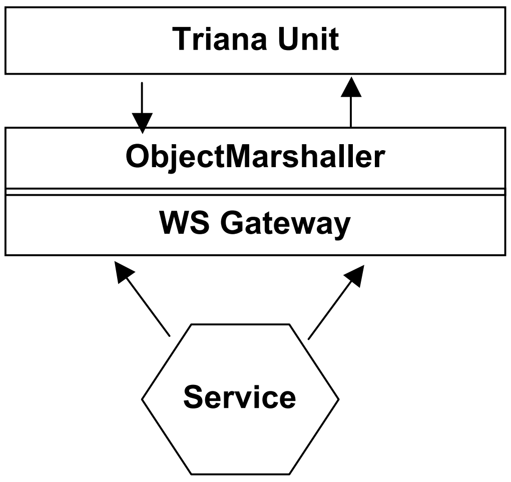

Figure 4: Service invocation

3.4 Publishing composite services

A user can make the composed service graph available to others. This is done by executing a simple GUI based wizard which asks the user to provide the relevant information for e.g. network location, a text description etc. All the necessary artifacts are automatically generated and the service published. The "PublishToUDDI" wizard first uploads the graph to the user specified location. A Triana launcher service running at the server generates a Triana tool representing the composite service. Second, the WSDL document is generated based on the WSDL documents of the component services. Finally, the service details are published to the UDDI server.

3.5 Triana Web services

There are already over 100 tools available for the Triana problem solving environment. These tools, ranging from image to signal processing, were developed by the Triana user community. Unfortunately, these tools cannot be exposed as Web services without undergoing extensive code revisions. To avoid this, we propose to implement a tool wrapper which can handle interactions with Triana tools and provides an interface for it to be invoked from a service. The wrapper configures and invokes the tool. The tool applies the algorithm and passes the results back to the wrapper, which then passes them on to the service (see Figure 5).

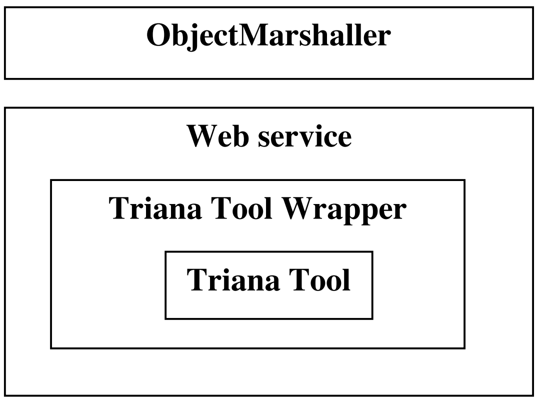

Figure 5: Wrapping Triana Tools

4.0 Conclusion

In this paper we have outlined a framework to integrate graphical creation of Web services workflows within the open source Triana problem solving environment. In particular, we looked at how Triana handles discovery, invocation, composition, and publishing of Web services through the WServe API. We have integrated UDDI within Triana enabling us to discover Web services. We have integrated BPEL4WS readers and writers which enables Triana to handle BPEL4WS graphs. We have developed a Web Services Gateway which enables Triana to execute Web services calls. Further, we have developed a facility which allows users to graphically and transparently publish composite services to the network.

Current and future work concerns the provision of data provenance, implementation of the complete BPEL4WS specification, integration with other workflow languages like Petri Nets. Additionally, we plan to extend this framework to include the OGSA services.

The Triana system is an ongoing project based at Cardiff University; the latest version can be downloaded at http://www.trianacode.org/.

5.0 References

[1] Fabio Casati, Ming-Chien Shan and D. Georgakopoulos, 2001, "E-Services - Guest editorial." The VLDB Journal 10(1): 1.

[2] A. Tsalgatidou and T. Pilioura, 2002, "An Overview of Standards and Related Technology in Web services.", Distributed and Parallel Databases, 12(3),

[3] D. Fensel, C. Bussler, Y. Ding, and B. Omelayenko, 2002, "The Web Service Modeling Framework WSMF", Electronic Commerce Research and Applications, 1(2).

[4] J.Cardoso and A.Sheth, 2002, "Semantic e-Workflow Composition. Technical Report, LSDIS Lab, Computer Science, University of Georgia.

[5] Ian Taylor, Matthew Shields and Ian Wang, 2003, "Resource Management of Triana P2P Services", Grid Resource Management, edited by Jan Weglarz, Jarek Nabrzyski, Jennifer Schopf and Maciej Stroinski, Kluwer.

[6] Ian Taylor, Matthew Shields, Ian Wang, Roger Philp, 2003, "Grid Enabling Applications Using Triana", Workshop on Grid Applications and Programming Tools, 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).

[7] Ian Taylor, Matt Shields, Ian Wang and Roger Philp, 2003, "Distributed P2P Computing within Triana: A Galaxy Visualization Test Case", IPDPS 2003 Conference.

[8] Gallopoulos, E., Houstis, E. N., and Rice, J. R., 1994, "Computer as thinker/doer: Problem solving environments for computational science", IEEE Comp. Sci. Engr. 1, 11-23.

[9] Satish Thatte, Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Ivana Trickovic, Sanjiva Weerawarana, 2003, "Business Process Execution Language for Web Services Version 1.1.", http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[10] K. Danzmann and the GEO Team, 1992, "The GEO Project: A Long Baseline Laser Interferometer for the Detection of Gravitational Waves", Lecture Notes in Physics 410 (1992) 184-209. http://www.geo600.uni-hannover.de/

[11] G. Allen et al., 2002, "GridLab: Enabling Applications on the Grid", Grid2002, 3rd International Workshop on Grid Computing, held in conjunction with Supercomputing 2002. Published as LNCS 2002 Vol. 2536, Pages 39-45

[12] I. Foster et al., 2002, "The physiology of the grid: An open grid services architecture for distributed systems integration", Open Grid Service Infrastructure WG, Global Grid Forum

[13] Project JXTA, 2003, http://www.jxta.org/