Workflows Hosted In Portals

Andrew Harrison†, Ian Kelley†‡, Kurt Mueller†, Matthew Shields†, Ian Taylor†‡

Abstract—The WHIP (Workflows Hosted In Portals) project is building a set of software plug-ins to enable interactive, user driven, workflow integration, sharing, and collaboration within Web portals. This software architecture enables the dynamic publishing of workflows, facilitating information exchange and scientific collaboration. The WHIP plug-in provides functionality to perform composition, editing and community publication of workflow descriptions and their associated semantics. In this paper we describe the WHIP architecture and provide an example use case through pairing the Triana workflow environment with the GridSphere portal using WHIP's client- and server-side APIs.

Index Terms—WHIP, OMII, Web Portals, Workflow, Collaboration, Information Exchange, Metadata

1 Introduction

Over the past several years, there has been much research and implementation undertaken in the use of workflow technologies for e-Science applications. In some cases, the activity has been driven by specific application domains, from supporting scientists in conducting *in silico* experiments in biology for Taverna, to gravitational wave experiments which have been the main underlying motivator for Triana. Other systems have been driven from a range of user communities and their requirements for large-scale science applications, such as Kepler, or Pegasus, for executing applications and moving data across Grid resources.

Most workflow engines employ the use of a customised user interface, which is not very well suited for direct integration into the more conventional user view of Grid computing, such as a Web portal interface. There are a number of problems with such a direct-integration approach, due to the complexity of the graphical user interface design and the often tightly-coupled dependencies to auxiliary information, such as task graphs, local processing requirements, and so forth. Further, such an approach is not scalable as each workflow system would need to implement its own mechanisms for achieving such an integration.

Portals, on the other hand, typically provide a framework for integrating information, applications, and processes in a customisable manner for its users through a ubiquitous desktop client, the Web browser. However, historically, such environments have been driven by users wishing to submit single jobs and therefore do not comprehensively address how dependencies can be represented and formalised between tasks for the creation of automated workflows. It is clear that some convergence between the workflow and portal interfaces needs to be undertaken in order to represent the broad application requirements of today's Grid computing communities but at the same time, we should avoid re-implementing such tools from scratch.

One possibility is to provide tools that support loosely-coupled mechanisms for supporting the coexistence of portal and workflow systems for e-Science applications. In this paper, we discuss one mechanism that is being developed within the OMII funded Workflows Hosted In Portals (WHIP) project. WHIP is providing a set of software plug-in utilities that enable the interaction between workflow, other desktop environments, and Web portals. The goals follow the ^{my}Experiment paradigm[1] allowing scientists to share their research in a workflow-centric, collaborative and Web portal-enabled environment. To enable this, WHIP is building plug-ins for portals and workflow environments that enables pairing and building of workflow–portal companions that can exchange stateless and/or stateful data. The pairing of the Triana workflow environment with the GridSphere Portal Framework, Taverna with ^{my}Experiment, and the BPEL-based Sedna workflow environment with the RAVE community[2] are target audiences of the WHIP project.

WHIP builds on this plug-in architecture and technology pairing to engage a wider community, applying itself to applications such as distributed visualisation and music information retrieval. The resulting components will enable a variety of workflows, in varying languages, for different workflow systems to be exposed and executed. Taverna, Triana, and BPEL workflow instances will be accessible from the portal, exposing a common model for further deployment into other research communities.

The primary goal and motivation of the WHIP project is to enable the sharing of complex artefacts in ways that have not been achieved before. Workflows are a good example of artefacts that are inherently complex: they are composites; they represent an activity – not a static thing; and they may require both local and remote processing

†*School of Computer Science, Cardiff University*
‡*Center for Computation and Technology, Louisiana State University*
Contact Ian.J.Taylor@cs.cardiff.ac.uk

1. http://www.myexperiment.org/
2. http://www.wesc.ac.uk/projectsite/rave/index.html

units. As a result, making such an object available to others in a coherent manner presumes access to a variety of diverse data by other users, for example executable code, processing data required by the workflow to run, provenance information, which allows a more rounded understanding of the workflow in its previous incarnations, particularly if the workflow contains remote invocations, as well as results from previous runs. Layered on top of this are concerns that relate, not directly to the understanding and enactment of the workflow, but to the community in which these are being shared. The context of this community provides further metadata that can be associated with the workflow, for example tagging, exposing related publications, and syndication of data. While these types of community-oriented interactions have become common-place in Web 2.0 environments, they have to date not been applied to objects with such a complexity as described here.

The rest of this paper is organised as follows. In the next section, we discuss the background to WHIP, followed by a discussion of the specific technologies we use to demonstrate how WHIP can be used to pair a portal with a workflow system by using Triana and GridSphere. In Section 2, we discuss related work in this area. In Section 3 we introduce WHIP and provide a comprehensive overview of its design and current implementation. Section 4 provides a use case showing how we have used WHIP to pair Triana as the workflow client with GridSphere as the portal implementation, Section 5 discusses other applications that will be integrated as part of the demonstrators for the project. Finally, we discuss the current implementation status and the conclusions for this work.

2 Background & Related Work

The WHIP project emerged from research conducted as part of the CoreGRID Network of Excellence [1]. This research was directed at integrating workflow clients and Web portals to enable the exposure of legacy codes in a Grid environment. A particular use case of this research was the ability to visually compose and run legacy components within the Cactus Computational Toolkit [2] as workflows, and combine this with the collaborative capabilities provided by the Cactus Portal [3], a Web interface to Cactus providing components for tracking simulations and sharing parameter data. Many Cactus applications are implicitly workflow based, for example, a complex fluid dynamics application may need an initial mesh to be constructed and fed into the simulation. The explicit modelling of Cactus applications as workflows allows the process to be managed in a simpler and more coherent way: Firstly, visual representation gives the user a clearer picture of the process. Secondly, the process can be stopped and started at cut points within the workflow, and thirdly, externally available computation, for example Web services providing solver functions for the calculation of the mesh, can be easily plugged into the workflow.

One of the results of this work was the first steps towards defining Web services interfaces for exchanging arbitrary data and metadata between Web based portals and thick clients. The WHIP project is developing these initial ideas and delivering a suite of components that can be used in a variety of contexts including service-oriented and resource-oriented environments, and both client/server and Peer-to-Peer topologies.

Workflow systems are well suited to Grid computing and can provide graphical environments that are an abstraction above Web portals, which typically deal with single *jobs* or simple *chains of jobs*. Adding such a facility to a portal would significantly enrich its capabilities for Grid users and would naturally fit in with common existing functionality, such as job tracking and status. The combination of portals and workflow systems is a natural progression in the Grid evolution as users strive for more demanding scenarios and more complex interactions between applications or services. Further, workflow support within such interfaces will become a necessity once applications progress to supporting more dynamic scenarios where logic control, perhaps based on data, can determine the particular workflow, or sub-workflow, being executed.

2.1 Workflow Systems

There are a number of workflow systems available which support Grid computing through specific techniques, allowing users to visually construct complex workflows of jobs or processes and then execute those on computational Grids. This is achieved through graphical environments often with diverse local and remote services and resources modelled as components, which can be dragged, dropped, and connected on a work surface, for a state-of-the-art overview of workflow systems see [4].

Workflow languages are the representation of the components or processes that are connected on the workspace. These languages are often described in XML dialects, modelling the cyclic or acyclic graphs which define the connections and dependencies.

Triana [5] started as a quick-look data analysis system for gravitational wave signal processing, now a general purpose scientific workflow tool, with respect to service-oriented architectures (SOA), Triana facilitates the publish, find, bind triad of operations that enable the impromptu assembly of distributed software assets through its GAP interface [6]. Through the GAP, Triana can interact with P2P Services using the JXTA and P2PS [7] bindings, Web services via WSPeer [8], and the Grid Application Toolkit (GAT) [9].

Taverna [10] originally based their components on Web services, although it now supports other types of transport protocols such as the Styx protocol [11], and includes support for local components. The Taverna workbench has been developed as a workflow front end to support *in silico* experiments in biology within the ^{my}Grid project. Taverna allows users to construct, share

and enact workflows consisting of services and components, using the ^{my}Grid Simple Conceptual Unified Flow Language (SCUFL), an XML based workflow language, to represent its workflow.

The Kepler project [12] for supporting scientific workflows is a cross-project collaboration based on the PtolemyII system [13]. The approach in Kepler/PtolemyII is very similar to Triana. Workflow is visually constructed from Java components, called actors. These components can invoke local processes or remote services such as Web services or a GridFTP transfer.

OMII-BPEL [14], funded as part of the UK based OMII Managed Programme, is a workflow tool based on the de-facto standard workflow language for business, BPEL [15]. The project incorporates a version of the ActiveBPEL[3] workflow engine and a graphical editor, Sedna, based upon the Eclipse platform.[4] Using a common standards based workflow language obviously has its advantages but opinion is divided upon whether the business domain based BPEL language can be effectively used for scientific workflow.

Pegasus [16] is a data driven workflow mapping engine that automatically maps high level scientific workflows to computational Grids. It allows scientists to define workflows in abstract terms separately from the infrastructures it will be deployed on, automatically managing data generated throughout the workflow execution.

2.2 Portals

GCEs have traditionally been implemented as Web portals, which provide access to Grid services and resources through a Web browser. Usually no further client-side software is needed to access a portal, which makes application deployment easy for developers and application access simple for users. However, the interactivity and interface richness of Web portals are somewhat limited in comparison with stand-alone desktop applications due to inherent limitations of Web browsers and Web protocols. Desktop GCE applications such as Triana, Kepler and Taverna provide a more full-featured and interactive environment than Web portals, often incorporating drag-and-drop user interfaces and deployable components, but their usage requires installation and configuration of software packages on client systems. Desktop applications provide an increase in functionality and sophistication at the expense of simplicity and ubiquitous access.

The Gridsphere Portal Framework [17] is a JSR-168 [18] compliant *portlet* hosting environment, or container. Portlets have established themselves as a way to componentize Web portals into their individual elements. For example, a portal could have independent portlets for user management capabilities, but also portlets ranging from news readers and stock tickers to Grid job execution and workflow management.

In addition to hosting user-created third-party portlets, GridSphere includes many additional built-in features such as advanced tag libraries, and portlets for content and user management, as well as a special class of portlets for Grid interaction, which are available through its companion project "GridPortlets" [19]. GridSphere is widely used in Europe, having its roots in the European-funded GridLab project. The GRIDCC[5] and P-GRADE[6] portals are examples of European projects currently using GridSphere. Notable GridSphere-based portals in the USA include the TeraGrid user portal[7], the BIRN portal[8], the SCOOP portal[9], and the CCT portal[10]. Further details about portlet standards and our use of GridSphere are provided in §4.2.

Recently there have been efforts to bring the power of desktop workflow applications to portals. The P-GRADE portal [20] has extended the concept of the traditional Grid portal to enable users to perform workflow functions. Through these additional workflow-enabling mechanisms, a user of the P-GRADE portal can construct, modify, archive, and share workflow, all without leaving the portal to install external software or relying on complex third-party applications. This is done through a combination of portal components accessible through a Web browser, and a Java Web Start application that can be launched from the portal. This combination allows for a more powerful client-side creation of workflow which is later sent to the portal for processing. However, compared to the functionality provided by Triana or Kepler, for example, the workflow creation and editing aspects offered by the P-GRADE portal are rudimentary, being limited to applications based on file inputs and outputs.

3 The WHIP Modules

WHIP is a set of lightweight components that can be used in a flexible way to achieve workflow sharing between diverse entities such as Web portals, workflow enactment engines, and Web services. While the primary development domain of WHIP is workflow sharing, it is capable of more generalised *artefact* sharing. Here, *artefact* refers to anything that is either output or input to some process or application and could include job or resource description documents, executable binaries, and dependent data. The modular nature of WHIP means the various sub-systems that make up the complete suite of capability are designed independently within the WHIP core framework and can be integrated incrementally, both in the portal and the client application.

3. http://www.activebpel.org
4. http://www.eclipse.org/
5. http://www.gridcc.org/
6. http://www.lpds.sztaki.hu/pgportal/
7. https://portal.teragrid.org
8. https://portal.nbirn.net
9. http://portal.scoop.sura.org/
10. http://portal.cct.lsu.edu

The aim of WHIP is to provide a simple, composable means of integrating a variety of software components with a view to enabling artefact sharing between them. The main actors in the artefact sharing process are:

- Web-based portals viewed through a browser interface. These store and present views of artefacts.
- Client applications that make use of these artefacts.
- The human user. Because of the nature of Web-based portals, the human user is a primary actor in the scenarios because all interaction with the portal is driven by their actions.

For example, a typical scenario might involve a user searching a digital signal processing portal looking for a matched filtering workflow, finding an appropriate example and selecting the link , which would launch the user's desktop workflow environment and load the example workflow. From the application side, a user may wish to submit an example of a particular workflow to a collaborative library of examples contained in a Web portal, and the application contacts the user portal, authenticates the user and uploads the workflow to the shared library. This example is examined in more detail in Section 4.

Beyond providing portal to client application communications, WHIP also enables communications between components within a single portal environment, and supports inter portal communications as well. These communications will usually not require nor benefit from human intervention. The goal of WHIP is therefore to integrate these different communication scenarios into a single modular framework.

WHIP is composed of two main modules. On the one hand, application independent interfaces to both data and metadata are defined which provide an entry point for arbitrary applications and services to integrate with WHIP plug-ins, and on the other, a series of components are being implemented that adhere to the interfaces, providing lightweight out-of-the-box capabilities. The following sections describe the metadata and data description interfaces, the implemented components that make use these interfaces, and an illustration of how they can integrated with existing applications.

3.1 Metadata and Data Module

Sharing artefacts requires that context, or metadata, about the artefacts can be defined and accessed. The metadata about an artefact should be accessible independently of the artefact itself, for example to allow intelligent reasoning about something before actually receiving it, thus saving bandwidth, or to restrict access to the actual artefact while still enabling it to be catalogued. Under these circumstances, the metadata should be able to provide a reference (location) from which the data can be drawn. On the other hand, metadata should also be able to travel with an artefact, allowing the recipient to understand the nature of the artefact if the metadata has not been viewed before. Furthermore, the metadata associated with an artefact is highly dependent on the domain from which that artefact stems, and the environment in which it is being transferred. WHIP provides a generic data description interface allowing access to metadata in a number of ways (e.g. through downloadable data archives or Web services invocations) while allowing domain specific applications and data stores to integrate their own metadata requirements.

WHIP defines a metadata document described in XML Schema. This document provides a means of:

- Exposing standard properties of data such as a unique identifier, author, owner, mime-type, creation time, and others.
- Flexible means of defining how data can be retrieved, for example using Web services standards such as WS-Addressing, or transfer protocol specific access mechanisms such as HTTP GET.
- Ability to reference external metadata.
- The inclusion of metadata specifications, e.g. Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH), the Data Format Description Language (DFDL) emerging from OGF and WS-MetadataExchange, either physically within the document, or referenced. This allows data that has already been catalogued according to an existing specification to be used by WHIP components.
- Identification of authorisation mechanisms, if data is referenced using transfer protocol specific means. (Web services interactions can use existing specifications for these purposes).

WHIP also defines an archiving format for bundling metadata and data together, based on the Java Archive (JAR) file format. This allows components to extract the metadata from an archive, and thence locate the data itself. The JAR format also allows components to sign and verify the signature of data archives. More details on these interfaces can be found at the WHIP Web site[11].

3.2 Plug-in Module

The WHIP plug-ins are written in the Java programming language in order make integration with existing target software (e.g. Taverna and Triana) easier, as well as to simplify embedding components into other environments. The WHIP portal plug-ins are implemented in accordance with the JSR-168 Java Portlet specification, described earlier.

JSR 168 has a number of limitations particularly with respect to the capabilities required by WHIP. Many of these limitations are being addressed by the second portlet specification (JSR-286); however, this specification is still in draft form, and there are no usable compliant implementations available. In particular the new specification addresses how portlets communicate with one another and how they serve resources, which are two key activities required by WHIP. Until such time as the

11. http://www.whipplugin.org

specification has reached maturity and is widely implemented, environment-specific solutions will be required for some of these activities.

To minimise the configuration required to integrate the browser interface with the client application, WHIP makes use of Java Web Start (JWS) technology. JWS is capable of parsing an XML configuration file and launching an application on the user's desktop based on the configuration parameters. JWS provides a number of features that are useful from WHIP's point of view:

- JWS provides a zero-configuration bridge between the browser environment and the application environment.
- JWS uses Java sand-boxing techniques for security. For example, all downloaded data must be signed, and accepted by the human user. The use of certificates in the download process forms a useful framework for implementing higher level authorisation policies.
- JWS can be configured by the user, for example to always trust applications or resources signed by particular entities. This reduces the need for user intervention if so desired.

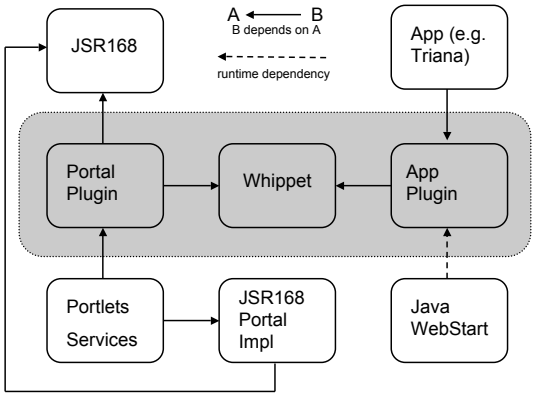

Fig. 1. WHIP Module Dependencies

JWS is part of the current deployment environment and not integral to WHIP components. Figure 1 shows the dependencies between the WHIP modules. The greyed area shows components provided by the WHIP plugins, whilst other components are third party. At the center is the *Whippet* package. This has no dependencies beyond the Java SDK 1.5. The *Portal Plugin* relies on the core *Whippet* library and the JSR 168 specification. The *Application Plugin* relies on the core *Whippet* library and has a runtime dependency on JWS; that is, it requires JWS to be instantiated, but does not contain references in the code to JWS. Applications using WHIP need only know about the *Application Plugin*, which has a very small and simple interface. Portlets and services running in an instance of a JSR 168 compliant portal, will have dependencies on that environment. For example, a persistent, singleton service is required to exchange messages between portlets, and this must be exposed according to the interfaces defined by the portal, because these capabilities are not defined in JSR 168. The code provided within the *Portal Plugin*, however, makes this very easy to achieve.

The core component in the *Whippet* library is the *Whippet* component. *Whippets* are essentially hash tables that can store key/value pairs locally, as well as retrieve and send key/value pairs to remote *Whippets* (or anything else that shares protocols supported by the *Whippet*). Hence *Whippets* support the operations associated with hash tables (put, get), in both a local and remote capacity. The *Whippet* package is designed to be very lightweight and easily embedded into a variety of applications. It therefore has no external dependencies beyond the JDK, and the jar file containing the code is roughly 70kb in size. Currently *Whippets* support HTTP for sending and receiving remote key/value pairs. Each *Whippet* acts as both a HTTP client and, if requested by an application, a micro HTTP server. Application code interacts with a *Whippet* via the *WhippetListener* interface, which receives notifications of events occurring on the *Whippet's* hash table.

The plug-ins make use of the *Whippet* package and add functionality on top. The *Portal Plugin* uses the local storage mechanisms for caching topics and messages shared between portlets. The remote putting and getting of the *Whippet* is used to perform the transfer of artefacts. At the client side, the HTTP functionality is used to communicate with the portal, and the local operations are used in communication with the application.

The *Portal Plugin* uses the core API to enable values to be sent between portlets and to remote client applications. As stated in Section 3.2, under JSR 168, portal environment specific services have to be written to provide portlet to portlet communication because there is no standard means of achieving this. The *Portal Plugin* defines a *WhippetService* interface, and an implementation, that simplifies this process.

The *WhippetService* allows portlets to discover what topics it is interested in publishing and consuming. These are defined in the *portlet.xml* file preferences elements. The *portlet.xml* file is part of the JSR 168 specification. Using the service's two methods – *getMyProducerPrefs(PortletRequest)* and *getMyConsumerPrefs(PortletRequest)*, portlets can retrieve these topics, and then either store or retrieve values associated with these topics with the service, using the *put* and *get* methods defined by it. These put and get methods exposed by the service, call back to an underlying *Whippet* that performs the storage and retrieval of values.

To facilitate communication between a portal and a remote client application, the *WhippetService* interface defines an operation that allows portlets to receive the URI of a data archive. Given a primary key for an object, the service returns a URI to the portlet. The portlet can then display this URI (e.g. using HTML or JSP), allowing

users to download data from it.

3.3 Java Web Start and the Portal Plugin

The current implementation of the *WhippetService* returns a Java Web Start (JWS) XML configuration file (*.jnlp*) from a PortletRequest for a URI. Hence, when the user clicks on the URI link in the portlet, the browser launches JWS on the local machine. This in turn reads the XML file, and pulls down the JAR files from the *Whippet* used by the *WhippetService* using HTTP. This download contains two JAR files, that are formatted according to the WHIP archive format (see Section 3.1). The first contains executable code to be launched. The second contains the data requested by the user. JWS checks with the *WhippetService* when the JAR files were last modified, and uses local copies (if available) if the JAR files have not changed. These JAR files are likely to be independently signed, (the application JAR by the authors of the executable code (a component of the *Portal Plugin*), and the data JAR by the owner of the data, or the portal). The user is asked if they trust the certificates of the signed JAR files. The user can reject the certificates, run them on a per-use basis, or choose to trust the certificates always. If the user decides to trust the data, then the *Portal Plugin* component in the application JAR is launched. This component looks for the metadata document defined in the data archive specification, and accesses the data object referenced by it. This data is then passed to a locally running *Whippet*. This locally running component is part of the *Application Plugin*, described in the next section.

4 Use Case

In this section, we describe how WHIP can be used to integrate the Triana workflow environment with GridSphere. We first discuss the Triana application perspective, and then discuss the GridSphere portal environment that is providing access to the application.

4.1 Triana Integration

The *ApplicationService* is an application's primary interface to the *Application Plugin*. This component launches a *Whippet* on the local machine at initialisation time. The *ApplicationService* has a *WhippetListener* associated with it. This interface defines two methods – *artefactArrived(String key)* and *getName()*, and is implemented by the application. The first of these allows the *ApplicationService* to notify the application of data arriving. The second is not currently used, but is intended to allow a listener to stay in contact with a *Whippet*, should the *Whippet* migrate to another host. In this case, the value returned from the *getName()* method, should return a network address.

The *ApplicationService* defines three methods – *put(String key, Object value)*, *get(String key)* and *destroy()*. The first of these allows an application to send an artefact. The second allows an application to retrieve an artefact after being notified of its arrival. The third method allows the application to close down the *ApplicationService*, and hence its underlying *Whippet*. Triana has an embedded *ApplicationService*, that is launched when Triana initialises. This service listens on a particular port, and notifies Triana when a Triana task graph arrives. On being notified, Triana retrieves the task graph from the service, and opens it on the work surface, for the user to inspect and run the workflow.

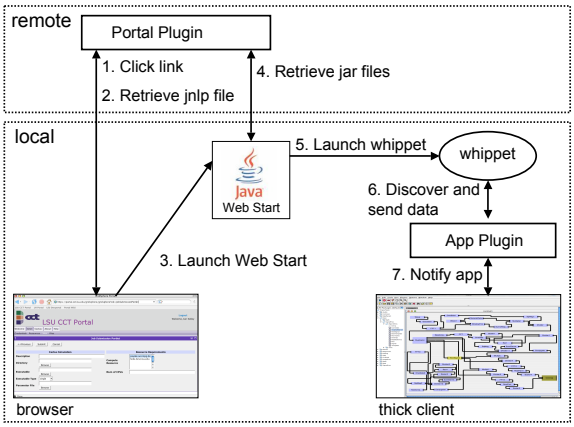

Fig. 2. WHIP Messaging Process

Figure 2 shows how the various components fit together to facilitate this. Seven stages are shown:

1) The user clicks on a link in the portal interface.
2) This points to a JNLP configuration file which is downloaded.
3) The browser launches Web Start, passing it the JNLP file.
4) Web Start downloads the jar files described in the JNLP file from the *Portal Plugin*.
5) A small *Whippet* is launched by Web Start.
6) This locates the appropriate data, which in the case of Triana is a packaged task graph, and looks for a locally running *Whippet* – the component embedded into Triana, and sends it the task graph.
7) Triana receives notification of the task graph's arrival via the *ApplicationService*.

4.2 GridSphere Implementation

The portal-side implementation of the *WhippetService* is very simple. It is currently loosely bound to GridSphere, as it implements the *PortletServiceProvider* interface defined in the GridSphere framework. Instances of these components are discovered by the framework when it initialises, and are thence accessible by portlets running in different applications. The *PortletServiceProvider* interface defines two lifetime-related methods – *init* and *destroy*, that are called by the framework at start-up and shut down of the portal. The *WhippetService* defines

analogous operations – *initWhippet()*, and *destroyWhippet()*, and hence the GridSphere specific service of the *WhippetService* implementation provided by WHIP simply calls the *WhippetService* lifetime methods from the *PortletServiceProvider* methods.

For a portlet to retrieve the *WhippetService*, portal specific means are also required. GridSphere provides a factory interface for retrieving PortletServices. Hence, accessing the GridSphere implementation of the *WhippetService* might look something like this:

```
PortletServiceFactory serviceFactory =
  SportletServiceFactory.getInstance();

WhippetService whip = (WhippetService)
  serviceFactory.createPortletService
  (WhippetService.class, true);
```

The Portlet can now retrieve its preferences defined in the *portlet.xml* file, and put and get values from the *WhippetService*.

5 WHIP Applications

As stated in Section 3, WHIP is suited not just to workflow sharing, but more general artefact sharing and will therefore be used in a variety of contexts. An existing OMII-UK use case is the sharing of Job Submission Description Language (JSDL) documents.

Within the context of workflow specifically, WHIP will initially be used to support the following scenarios:

The Taverna workbench was tailored for life scientists. To date, the sharing of Taverna workflows and workflow results has been done in a haphazard and unsystematic way. We will showcase how WHIP enables linking the Taverna editor to the ^{my}Experiment portal, allowing Taverna users to selectively publish and share workflows and workflow runs in a shared space.

The Distributed Audio Retrieval using Triana (DART) project [21] provides a decentralised overlay for processing audio information through its Music Information Retrieval (MIR) mechanisms. In DART, users provide CPU cycles for analysis of their local audio files, which in turn provides audio metadata to the network that is then used to enable the system to make music recommendations based on collaborative filtering and music information retrieval techniques. The local processing on the network participants is achieved though the use of Triana workflows that can be uploaded to the members in the network. In this scenario, scientists will be able to download and execute workflows, which will involve sharing the workflow as well as the Java code (and versions) in the form of a workflow package. WHIP will provide the mechanism enabling this type of collaboration.

A RAVE workflow, in general, is any workflow that includes a RAVE node for collaborative visualisation. Clicking on an active RAVE node brings up a portlet that shows if there are any collaborative visualisation sessions currently accessing the data passed to RAVE in the workflow instance. Users can choose to join an existing session or initiate a new one (which will then be visible to any other users who subsequently click on the RAVE node). The RAVE Portlet supports a plug-in mechanism that allows the conversion, if necessary, of the data input to the RAVE node to a form recognised by RAVE. A portal for sharing RAVE-enabled nanoCMOS workflows constructed with the SEDNA eclipse plug-in will be enabled though allowing its users to share, manage and execute their workflows.

6 Project Status

The APIs and schema defined by WHIP are prototypical and are a first step to developing a sophisticated artefact sharing infrastructure. The next phase of development will focus on:

- Integrating Web services capabilities into the whippet API. This will enable more complex exchanges, tracking of resources, authorisation policy descriptions and portal to portal metadata exchange.
- Developing a full-featured GridSphere portal based on the whippet API that enables sharing and management of Triana workflow components by Triana application and portal users, including authentication of data and authorisation of users, and tracking of workflows as they progress.
- Further plug-ins allowing sharing of WHIP defined data and metadata under different conditions. Example plug-ins under development are Atom Publishing Protocol [12] client- and server-side modules allowing users to receive notification of living documents and provide syndication mechanisms for portals. A plug-in targeted at Servlet containers is also under development.

While any new technology has potential barriers to adoption, WHIP is specifically focussed on making it easy for users, in the form of scientists, and applications and containers to make use of its capabilities. Firstly both SOAP based, and REST oriented communications are supported. The REST approach provides a very low entry point for client applications to send and receive data. Secondly, the core WHIP library is small and has no external dependencies beyond a Java 1.5 JRE, which means application dependencies are not bloated. Thirdly, WHIP provides out-of-the-box tools for parsing and packaging WHIP defined artefacts.

A first release of the software will be announced in the September of 2007, coinciding with the formal publishing of a WHIP Web site providing bug and issue tracking, source and binary downloads and documentation.

12. http://bitworking.org/projects/atom/draft-ietf-atompub-protocol-15.html

7 Conclusions

In this paper we have introduced the WHIP project, describing the motivation for integrating workflow enactment and sharing, with Web portal technologies to provide a collaborative, community environment where scientists can share workflow, metadata and other Grid computing resources in an application agnostic manner. The data and metadata interfaces were introduced, and the initial plug-ins developed so far were described. We also described the use of Triana and the GridSphere portal framework as an example pairing, and how other workflow application/portal pairings can use the same technologies to build dynamic, domain specific, scientific user communities.

8 Acknowledgements

The authors wish to thank both the GridSphere and Triana teams for their comments and discussions. This work was supported by an OMII-UK grant and the Center for Computation & Technology at Louisiana State University.

References

[1] A. Harrison, I. Kelley, and I. Taylor. Workflow-Driven Portals for the Integration of Legacy Applications. In *Grid-Enabling Legacy Applications and Supporting End Users Workshop (GELA) within the framework of the 15th IEEE International Symposium on High Performance Distributed Computing, HPDC'15*, June 19-23, 2006.

[2] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf. The cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing VECPAR 2002, 5th International Conference, Lecture Notes in Computer Science*. Springer, 2003.

[3] Ian Kelley, Oliver Wehrens, Michael Russell, and Jason Novotny. The cactus portal. In *Proceedings of APAC 05: Advanced Computing, Grid Applications and eResearch*, 2005.

[4] Ian Taylor, Ewa Deelman, Dennis Gannon, and Matthew Shields (Eds.). *Workflows for e-Science*. Springer, New York, Secaucus, NJ, USA, 2007.

[5] David Churches, Gabor Gombas, Andrew Harrison, Jason Maassen, Craig Robinson, Matthew Shields, Ian Taylor, and Ian Wang. Programming Scientific and Distributed Workflow with Triana Services. *Grid Workflow 2004 Special Issue of Concurrency and Computation: Practice and Experience*, To be published, 2005.

[6] Ian Taylor, Matthew Shields, Ian Wang, and Omer Rana. Triana Applications within Grid Computing and Peer to Peer Environments. *Journal of Grid Computing*, 1(2):199–217, 2003.

[7] Ian Wang. P2PS (Peer-to-Peer Simplified). In *Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies*, pages 54–59. Louisiana State University, February 2005.

[8] A. Harrison and I. Taylor. The Web Services Resource Framework In A Peer-To-Peer Context. *Journal of Grid Computing*, 4(4):425–445, December 2006.

[9] Gabrielle Allen, Kelly Davis, Konstantinos N. Dolkas, Nikolaos D. Doulamis, Tom Goodale, Thilo Kielmann, André Merzky, Jarek Nabrzyski, Juliusz Pukacki, Thomas Radke, Michael Russell, Ed Seidel, John Shalf, and Ian Taylor. Enabling applications on the grid: A gridlab overview. *International Journal of High Performance Computing Applications: Special Issue on Grid Computing: Infrastructure and Applications*, 17(4):449–466, November 2003.

[10] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics Journal*, 20(17):3045–3054, June 2004.

[11] J. Blower, A. Harrison, and K. Haines. Styx Grid Services: Lightweight, Easy-to-use Middleware For Scientific Workflows. In *First International Workshop on Workflow systems in e-Science - WSES06 in conjunction with International Conference on Computational Science (ICCS) 2006*, 2006.

[12] B. Ludäscher, I. Altintas, C. Berkley, D. G. Higgins, E. Jaeger, M. Jones, E. A. Lee, and Y. Zhao. Scientific workflow management and the kepler system. concurrency and computation: Practice and experience. *Concurrency and Computation: Practice and Experience, Special Issue on Scientific Workflows*, 2006. to appear.

[13] Ptolemy II. `http://ptolemy.eecs.berkeley.edu/ptolemyII`.

[14] B. Wassermann, W. Emmerich, and B. Butchart. Sedna: A BPEL-based environment for visual scientific workflow modelling. In Ian Taylor, Ewa Deelman, Dennis Gannon, and Matthew Shields, editors, *Workflows for e-Science*, pages 428–449. Springer, New York, Secaucus, NJ, USA, 2007.

[15] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services Version 1.1.

[16] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D.S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13(2), November 2005.

[17] J. Novotny, M. Russell, and O. Wehrens. GridSphere: A Portal Framework for Building Collaborations. In *1st International Workshop on Middleware for Grid Computing (at ACM/IFIP/USENIX Middleware 2003)*, 2003.

[18] The Java Community Process. Portlet specification. `http://jcp.org/en/jsr/detail?id=168`.

[19] Chongjie. Zhang, Ian Kelley, and Gabrielle Allen. Grid portal solutions: A comparison of gridportlets and ogce. *To be published in a special issue of Concurrency and Computation: Practice and Experience*.

[20] P. Kacsuk and G. Sipos. Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal. *Journal of Grid Computing*, 3(3-4), 2005.

[21] Ian Taylor, Eddie Al-Shakarchi, and Stephen David Beck. Distributed Audio Retrieval using Triana (DART). In *International Computer Music Conference (ICMC) 2006, November 6-11, at Tulane University, USA.*, pages 716 – 722, 2006.