

# ASToolkit: Web Service Wrapper for Scientific Applications

Y. Chen, A. Khanban†, G. Fagiolo‡, K. Saleem, J.V. Hajnal‡, D.L.G. Hill, D. Rueckert†

Department of Computer Science, UCL

Department of Computing, Imperial College London †

Imaging Sciences Department, Imperial College London ‡

## Abstract

Grid infrastructures have been used to achieve large-scale compute resource sharing and coordinated problem solving in dynamic multi-institutional Virtual Organizations. However, it is quite challenging to program and use these distributed infrastructures. In recent years, Service-Oriented Architecture (SOA) using Web services has gained growing attention. SOA has shown its enormous potential because it provides language-neutral service-interfaces that hide the complexity of the implementations. Since most scientific applications are not Web service oriented, and the adaptation of these existing scientific applications to Web services is becoming important along with the emergence of Grid and SOA. In this paper, we present an Application Service Toolkit (ASToolkit), which is a toolkit to wrap scientific applications as Web services. It provides features such as Grid-enabling, WS-Security, data management, and minimum software requirement.

## 1. Introduction

In order to solve complicated health-care, medical-research and drug-discovery problems, it is often needed to integrate various computationally-intensive image-processing algorithms and to customise workflows to suit a particular problem. For example, to enable quantitative comparisons between Magnetic Resonance (MR) brain images a batch brain-atlas registration workflow could be performed. This workflow could be composed of a brain extraction tool (such as FSL BET [1]), and a registration toolkit (such as IRTK [2]). These commonly used scientific applications have been developed by different teams of researchers and typically represent high quality and validated software. They are required to be integrated together to solve large-scale medical imaging problems.

Hospitals and medical research institutions usually have digital archives, with many of them holding terabytes of data. Currently these image repositories tend to be isolated from one another, with poor interoperability between them. Sharing imaging data in a secure and controlled way has the potential to add significant value to improve the quantitative medical investigations of clinicians and researchers.

To meet the challenges of both analysis-algorithms and data-sharing encountered in large scale medical research problems, NeuroGrid aims to provide a practical and flexible integration framework to the neuroimaging community. A Service-oriented framework built on a Grid infrastructure offers a

potential solution with the promise of distributed computational resource sharing, application sharing, and transparent access to distributed data repositories.

Non-grid distributed computing can certainly provide homogeneous, reliable, and secure computational power to solve computationally intensive problems. However, the Grid, as explained by Ian Foster and Carl Kesselman [3], should enable “resource sharing and coordinated problem solving in dynamic, multi-institutional Virtual Organizations”. Grid computing is akin to distributed computing, yet with a major focus on collaboration, data sharing, and other interaction on a global scale. Furthermore it allows heterogeneous computational resources to be deployed in a transparent way. Grid infrastructures have been used to achieve large-scale and coordinated compute resource sharing among dynamic collections of individuals and institutions.

Unfortunately, the programming and use of these distributed infrastructures is often quite challenging due to a multiplicity of hardware configurations, software configurations and administrative policies. Service-Oriented Architecture (SOA) has the potential to address this problem because it provides language neutral service interfaces that hide the complexity of specific implementations. SOA is an architecture made up of components and interconnections that stresses interoperability and location transparency. The idea of SOA is to achieve loose coupling among interacting software. This advanced flexible style of architecture provides the foundation to allow Grid resources to be shared seamlessly.

However, in most scientific fields, including medical image analysis, existing algorithms usually are not Web service oriented, and they are written in various programming languages (e.g. FORTRAN, C/C++, scripting languages and others). These codes not only typically represent large-scale investments in terms of time and effort that cannot be discarded, but also are high-quality and extensively validated programs. The adaptation of these existing applications to Web services is becoming important as a way of harnessing validated tools in a new powerful operational environment provided by the Grid and SOA.

The NeuroGrid project team has been developing the Application Service Toolkit (ASToolkit) to be able to wrap scientific applications with a Web service interface. The ASToolkit makes command oriented application wrapping an easy and fast task. In this paper the features and implementation of this Web service wrapper are described together with how it has been used and tested in the context of medical image analysis.

The paper is organized as follows. In section 2, the NeuroGrid project is introduced and the n-layer Service-Oriented Architecture is described. Section 3 introduces the features of ASToolkit and the technical details of its implementation. In section 4, three user cases are described, where medical image analysis algorithms are wrapped as Web services via the ASToolkit and composed as workflows. Finally, some concluding remarks are made.

## 2. NeuroGrid and Architecture

NeuroGrid [4] is an MRC e-science research project aimed at exploring the use of Grid technology for the neuroimaging community. The project participants include Oxford, UCL, Imperial College, Edinburgh, Nottingham, Cambridge, and Newcastle. The project aims: 1. To provide enabling image analysis capabilities for the neuroimaging community using Grid technology. 2. To allow current algorithms and existing data management procedures to be more accessible and interoperable. 3. To test the proposed solutions on clinical exemplars focused on Stroke, Psychosis and Dementia. The geographically distributed participants can publish and query data to and from global data repositories, specify the service configuration via a grid portal, remotely process medical image analysis algorithms on demand, visualise in real-time and collaboratively analyse the results.

Based on the project requirements, three major areas have been investigated: variability in the type of computational resources, variability in the applications and variability in the users. The NeuroGrid infrastructure consists of heterogeneous resources across widely geographically distributed Virtual Organizations. Any individual participant can have a variety of existing applications to contribute to the NeuroGrid. Moreover, NeuroGrid brings together four distinct roles: service consumer, service provider, service broker and computing resource provider. Each participant can play multiple roles. Figure 1 illustrates the relationship among these four roles.

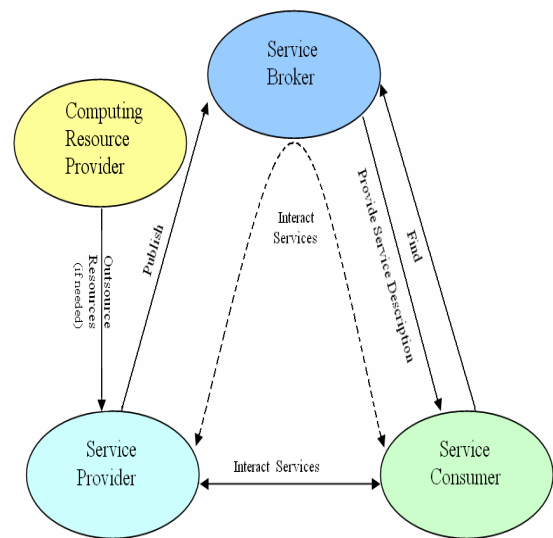
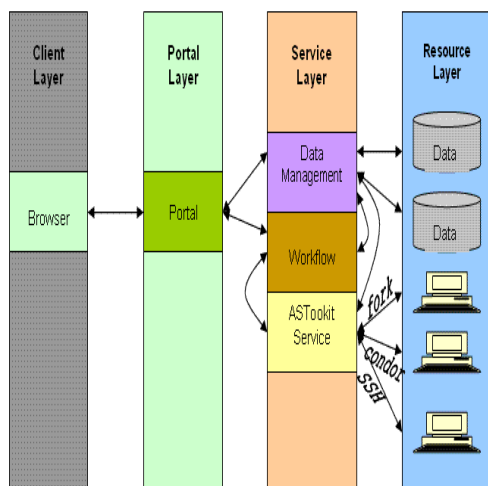


Fig. 1. NeuroGrid Roles and Relationships

The NeuroGrid architecture is designed to be modular and adaptable and is organized in multiple layers based on a Service-Oriented approach to promote interoperability and allow reuse of core components. This architecture is shown in Figure 2.



**Fig. 2.** NeuroGrid n-layer Architecture

SOA is adopted as shown in Figure 2, with a layered architectural framework including a client layer, a portal layer, a Web services layer, and a compute resources layer. The client communicates with the portal layer by sending and receiving SOAP over SSL, and utilizes the portal for core functionalities. The portal in turn forwards some of the incoming requests to a series of Web services and relies upon underlying Web services to provide functionalities to the client.

The client layer only consists of a compatible Web browser so that it is lightweight and can operate across firewalls. Any system that can run a Web browser can be a client for NeuroGrid. The portal layer provides an end-user interface, through which users can access data and algorithms services from anywhere with a browser and an Internet connection.

The portal is a user friendly Web-based interface that remotely launches and monitors medical image algorithms on NeuroGrid compute resources at remote sites. It also provides the interface to publish and query the image data to and from the global image databases. It provides seamless integration of a collection of medical analysis applications and medical data across geographically distributed virtual organizations. The portal presents the data and applications to the end-user through a browser and an Internet connection, and hides the user from the complexities of the underlying Grid infrastructure.

The business logic layer includes a collection of Web services including a data management service provided by the NeuroGrid Oxford team, and ASToolkit application services are provided by application providers. The applications are decomposed into component-

oriented services, which are exposed in a standard interface independent of the implementation languages and platforms. The service provider has the flexibility to move services to different machines, or to move a service to an external provider. Some services can support different client types. These applications, such as a pool of Web services, are the fundamental components of this architecture. Since Web services are platform and language independent, a Web service technology is also used to implement support services between applications and the portal. Some support Web services have been developed to provide the following functionalities: data transfer, data publishing and query, workflow management, job monitoring and others.

The resources layer includes the underlying computational resources and data storage resources.

### 3. Application Service Toolkit

#### 3.1 Overview

Mature software with known reliability and performance characteristics needs to be used for the NeuroGrid project. The project's primary consideration in developing the ASToolkit is to use standard software and keep the software requirements to a minimum.

ASToolkit is a toolkit that wraps scientific applications as Web services. ASToolkit can wrap almost any command line application (i.e. non-graphical), such as UNIX commands, or more sophisticated scripts written in Python, Perl, and so on. Most large scale computational facilities have traditionally operated their machines in batch mode and the ASToolkit is focused on these batch mode applications. It does not require any modification to the wrapped applications.

ASToolkit services are capable of serving multiple client requests concurrently. Also, since Web services are language and platform independent, they are easily accessible by clients written in different languages.

The Web service generated by the ASToolkit can enable but does not require the use of distributed resources via the Grid. The ASToolkit service can support operation seamlessly in a highly distributed environment. The distributed functionality is enabled and controlled by components employing Grid middleware. While the ASToolkit service can fully support the distributed environment, it also can be easily used in a local environment. The

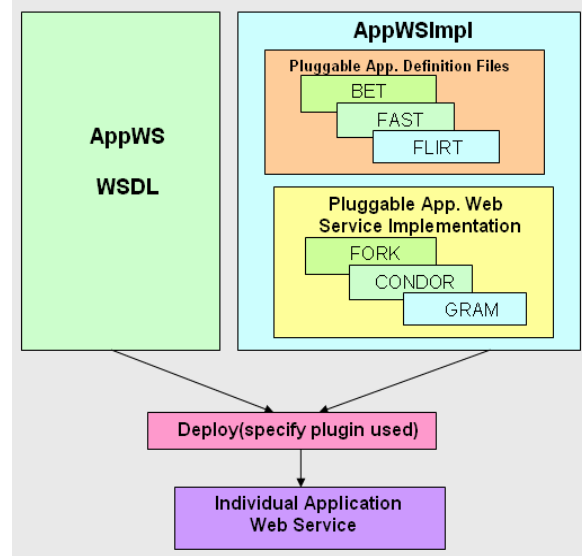
role of the Grid-enabled ASToolkit service is to provide a uniform submission layer on top of different execution environments.

### 3.2 Features Overview

The ASToolkit provides the following features:

- Easily and rapidly wrap any command oriented application as a Web service without modification to the selected application.
- Enable strong data typing. ASToolkit services have inputs and outputs defined in detail using XML schemas. Initially the number of types and a rich set of elements are declared. This ensures strongly typed data exchange among services. The strong data typing not only makes validation of inputs easy, but also facilitates generic Web service workflow tools to compose these services easily.
- Adopt consistent interface and plug-in model. All the ASToolkit services have the same interface and can be easily extended to support any Grid scheduling. ASToolkit services are not tied to any specific scheduling middleware.
- Hide the computational resource from the user. The user only interacts with algorithm services, not computational resources.
- Implement with WS-Security, which address the three security requirements: message authentication, message integrity and message confidentiality.
- Use a Service Provider Account Model to make user account management simpler. There is no need to open a user account for each user on computational resource.
- Support data transfer for jobs. ASToolkit services can retrieve/bring back the input/output data from/to user specified locations.
- Support concurrent and asynchronous job submissions.
- Support job monitoring.
- Have capability to return job specification provided by user for each job. This is useful to restore the job if the job fails.

### 3.3 Consistent Interface



*Application web service with well-defined interface*

**Fig. 3.** ASToolkit Service with Consistent Interface

As Figure 3 shows, all the ASToolkit services use the same interface no matter what standard application is wrapped. Disregarding how many applications are wrapped and deployed, what input data and options they expect, what output they produce, what syntax the command line applications have, they all use the consistent interface and are controlled by the same methods.

From Figure 3 we can also see that all ASToolkit services use the same interface no matter what grid scheduler they interact with, FORK, CONDOR, GRAM, or others. ASToolkit service is Grid enabled and it can submit jobs to distributed compute resources. Since various schedulers run on different sites, it is mandatory that these schedulers can be accessed in a consistent way for maximum code reuse. Furthermore, Web service implementations interact with these schedulers programmatically instead of via their regular command line interfaces.

This consistent approach can offer a lot of benefits for users only interested in the image analysis services themselves, but not in the underlying complicated compute resource, or the diverse scheduler middleware. By providing one consistent interface to all applications, we effectively hide the complicated grid environment from the user.

We use a static WSDL for every ASToolkit service due to the consistent interface. Via this approach, we avoid dynamically generating

source code and WSDL, which is usually error prone and is specific to the SOAP toolkits used. Two ASToolkit services can be distinguished by their unique URLs, and their associated Application Definition File.

### 3.4 Component Plug-in Model

Figure 3 also shows the component plug-in model we use for ASToolkit service. “AppWSImpl”, the business logic of ASToolkit service, is encapsulated behind the consistent interface of the service. Pluggable Application Definition Files and Web service implementations are prepared and plugged in at deployment time or run time in order to dynamically construct individual application services. The use of the component plug-in model ensures that our services are not tied to any specific middleware package. It is straightforward to implement other plug-ins to submit to other schedulers. This makes the ASToolkit service easy to fit in the increasingly diverse and complex Grid middleware

At present we have implemented two types of components, Fork and Condor. The fork implementation translates an incoming request combined with the Application Definition File, into a simple command line and forks this on the host server of the application service; while the Condor implementation takes the incoming request and programmatically interacts with the SOAP interface of Condor scheduler at a remote site.

### 3.5 Command Line Description Language

Command line Description Language (CoLDeL), an XML based language, is designed in order to describe individual applications precisely. CoLDeL acts as a standard so that different service providers could follow it to generate an Application Definition File for each scientific algorithm for use by ASToolkit Services. An XML schema has been defined which ensures strongly typed data exchanging among services. An initial number of types and a rich set of elements are declared.

The design of the CoLDeL is simple but powerful. It provides a generic approach to the abstraction of command oriented application’s configuration. It makes the application service lightweight but highly configurable. No business logic code generation is needed to create a service from its description. CoLDeL is also helpful to enable generic Web service workflow tools to compose application services.

In our current implementation, CoLDeL has following advantages:

- Conforming to the XML schema, CoLDeL can specify each application with a rich semantic description and provide as much useful information to the service/workflow/user as possible.
- CoLDeL provides a set of default arguments values from Application Definition File, so that the user needs not set all argument values for each job execution – this is useful as there can be hundreds of them to be set up, but the user is often only interested in few of them.
- CoLDeL supports data types and constraints on arguments values to ensure that all the arguments values are acceptable. It can facilitate validation of the user’s job configuration. The argument set up from the user can be validated against its data type and constraints before execution. This increases the probability of successful completion of the execution. The benefit of this validation is particularly obvious for long-running applications or for applications that form a part of a workflow.
- CoLDeL specifies the dependencies of arguments, which can help to validate the input dependencies.

Both client and server follow CoLDeL to specify and process the application description. At server side, before wrapping the application, the application provider must write an Application Definition File for every individual application, an XML document, that we call the CoLDeL document. The CoLDeL document has to be complete enough so that the service can dynamically compose the command line at run time and retrieve the input data files. As a consequence, the CoLDeL document contains following information that can be categorized into three categories:

- General information. This includes the algorithm name, contributing institution, versioning information, and brief description.
- The execution environment information. This describes the requirement on the execution environment such as platform, libraries required and environment variables. This information is used to construct the execution environment at run time.
- The arguments description. This provides all the information for each argument of the algorithm command line, including mandatory information, argument type,

default values, value range, dependency information, naming conventions of outputs, and so on.

On the ASToolkit client side, the client builds strongly typed data to supply specific argument values for the application through incoming an SOAP message. The ASToolkit service will take this message along with the CoLDel document, and dynamically construct the actual command line at the execution time. The input data arguments have to be set by user. The user needs set up the Uniform Resource Identifier of the input files. See next section for details about data management.

JAVA XMLDecoder and XMLEncoder are applied for converting a CoLDel object to and from its equivalent XML document representation. The ASToolkit service generates a CoLDel object from each CoLDel document, combines this with the CoLDel object provided by the ASToolkit client, determines the specifics on how to properly build and construct the command with the appropriate parameters, and subsequently submits the job.

### 3.6 Data Management

The ASToolkit service handles data transfer between service provider and data repositories. It retrieves the input data from the data repositories and puts the results back to the data repositories. Uniform Resource Identifier (URI), a compact string of characters for identifying an abstract or physical resource, is adopted to provide the direct link to data files and processing facilities. Thus, the information exchanged between the ASToolkit client and service via SOAP only contains references of input/output, the URIs. No large data is included into SOAP messages. Since we are using WebDAV [5] to manage data, The URI contains information of the WebDAV server, WebDAV folder name, and file name.

We believe using a common data transfer protocol would eliminate the current duplication of effort in developing unique data transfer capabilities for different storage systems. For the current implementation, the ASToolkit service only takes input data located at WebDAV folder on the remote data server. WebDAV is used to transfer data files between the ASToolkit service and data server, which is a secure, efficient data transport mechanism. If needed, other data management plug-ins are possible and can easily be fitted into the ASToolkit due to the component plug-in model used.

When a client invokes an application, the service provider creates a new working

directory for each job at the service provider site. The input data files described via URIs are archived from data repositories to the corresponding working directory through WebDAV. The application is run inside the working directory in the FORK case, or submitted to a computational resource. The output will be brought back to this working directory from the remote computational site. Finally ASToolkit service transfers all the output files back to WebDAV folder on the data server through WebDAV. The URIs of output files are provided which allow them to be located and retrieved later.

### 3.7 Security

Security is a critical requirement and must be accounted for by any geographically distributed Grid community. There is high demand to protect data confidentiality and integrity in neuroimaging applications. The traditional security mechanisms for homogeneous systems do not scale to heterogeneous environments operated by different organizations. A reliable yet easy to use security infrastructure is therefore important.

Due to stringent requirements for patient confidentiality and authenticity of results, we choose WS-Security [6] instead of transport level security to ensure secure data management in a distributed environment. Transport based security is bound to HTTP. It only secures the transport channel between two points. This solution is incomplete if intermediaries between the endpoints forward or process the message. While with WS-Security, the message itself is secure rather than just the underlying transport. Signature and encryption persists with messages. It also supports partial message signing and encryption.

The ASToolkit service with WS-Security addresses the three security requirements outlined below:

- Authentication is used to ensure the identity of the message senders.
- Digital signatures are used to ensure a message's integrity, that its content has not been altered or corrupted during its transmission over the network.
- Encryption is used to ensure message confidentiality.

In a typical usage scenario, the user invokes the service provided by service provider, and then the service provider accesses compute resources on behalf of user. This means compute resources have to trust all the users trusted by service provider, and hence brings the

account management burden to the resource provider. We introduce a Service Provider Account Model to avoid this problem. In this model, the user does not have direct access to compute resources and they are completely decoupled from compute resources where jobs are effectively run. The user only has access to the ASToolkit service. A special user account is set up for service provider, which is trusted by compute resources. The service provider acts as an active agent between the user and compute resources. It authenticates and authorizes the users, serves the user requests. It also accesses related compute resources on behalf of itself. The compute resources respond to these requests due to their trust of the service provider.

#### 4. User Cases

Several medical image analysis algorithms have been exposed as Web services easily and rapidly via the ASToolkit. The toolkit is also likely to be easily used for command oriented applications in other fields. As mentioned in section 3.3, with the focus on simplicity and configurability, all the ASToolkit services use a consistent interface and the same business logic. ASToolkit services can be distinguished by their unique URLs, and their associated Application Definition File. This approach makes wrapping easy but stable. The service provider only needs create one appropriate CoLDeL document, specify the application behavior (FORK or CONDOR) by another configuration file, deploy the application as aWeb service using simple a deployment mechanism (via Apache Ant).

Some medical applications are exposed as Web services via ASToolkit. In real world, imaging studies tend to utilise sequential pipelines of image processing algorithms where the results of one algorithm are used as the input to a subsequent step. These pipelines are defined by the researcher and then applied to specific data sets. The ability to compose ASToolkit services sequentially is an essential requirement. Based on this requirement, a practical workflow engine has been developed and used to create the sequential scientific pipelines. The workflow is expressed in a XML based workflow description file. The workflow engine co-ordinates the execution of a series of sequential ASToolkit services as specified in the workflow description file. This workflow engine allows user to submit sequential pipeline jobs in one go, and further monitor and restore

them if the job fails. It also has batch job submission functionalities.

To provide examples we created the following three simple scientific workflows.

#### 4.1 Brain Extraction and Segmentation Workflow

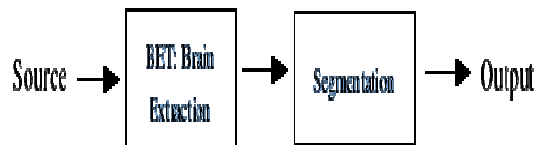


Fig. 4. Brain Extraction - Segmentation Workflow

As shown in figure 4, two algorithms wrapped as ASToolkit services are used in this segmentation workflow. One is FMRIB BET, Brain Extraction Tool; and the other is FMRIB FAST, automated tissue classification tool. This workflow segments the brain from MR image sets, removes surrounding and peripheral tissues, then classifies voxels into different tissue groups and returns a segmented image.

#### 4.2 Brain Extraction, Affine Registration and Transformation Workflow

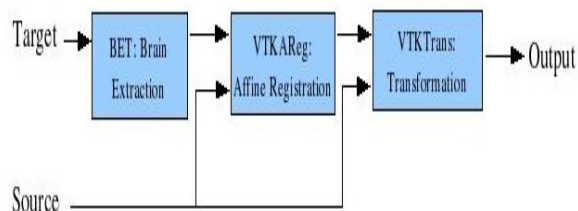
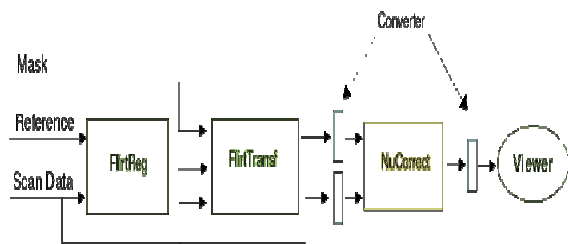


Fig. 5. Brain Extraction - Affine Registration - Transformation Workflow

Figure 5 shows that there are three ASToolkit services in this registration workflow. They are: brain extraction; affine registration; and transformation as determined by the registration process. This workflow automatically aligns a source image with a reference image. First, the brain extraction is used to eliminate non-brain tissue from the reference image, so that the registration is focused on matching the brain. Next the quality of alignment is determined by a similarity measure between the two images. The alignment uses 12 parameters affine transformation, 3 rotations, 3 translations, 3 scaling factors, and 3 skew factors. Finally, the input image is transformed according to the parameters computed by the registration step.

### 4.3 Image Intensity Correction Workflow



**Fig. 6.** Flirt Registration – Flirt Transformation  
- Intensity Correction Workflow

As Figure 6 shows, the algorithms wrapped in this workflow are FMRIB FLIRT - image registration, MNI N3 – intensity correction, and FMRIB converter – applies image transformation. The process is: Aligns mask with target image; estimates bias-field correction using N3; applies correction; returns corrected image.

Each of these workflows has been tested on sample MR images from the IXI data set ([www.ixi.org.uk](http://www.ixi.org.uk)) and found to operate effectively from a simple web page interface that can be used by non-experts.

## 5. Conclusions

In this paper, we presented ASToolkit, a toolkit for wrapping scientific applications as Web services. We described the technical details of the ASToolkit implementation, and demonstrated its successful usage on some medical image analysis algorithms. The Condor plug-in has been developed and tested on UCL Condor pool cluster. Our generic framework allows users to wrap applications as Web services, compose sequential workflows from them, execute them on a distributed compute resource, and monitor/restore the job in an easy to user manner. This framework has been released to researchers within the NeuroGrid consortium for further testing and use within exemplar research projects. In future it will be enhanced based on the user feedback.

## References

- [1] S.M. Smith. Fast robust automated brain extraction. *Human Brain Mapping*, 17(3):143-155, November 2002.
- [2] Rueckert D, <http://www.doc.ic.ac.uk/~dr/software>
- [3] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the grid: An open

grid services architecture for distributed systems integration, June 2002.

[4] John Geddes, Sharon Lloyd, Andrew Simpson, Martin Rossor, Nick Fox, Derek Hill, Joseph Hajnal, etc. NeuroGrid: Collaborative Neuroscience via Grid Computing. UK e-Science All Hands Meeting, 2004 Nottingham, UK.

[5] Jim Whitehead, Yaron Y. Goland. WebDAV: A network protocol for remote collaborative authoring on the Web. European Computer Supported Cooperative Work (ECSCW'99)

[6] Web Services Security (WS-Security) <http://msdn2.microsoft.com/enus/library/ms951257.aspx>