



**Bilateral Research and Industrial Development Enhancing and Integrating
GRID Enabled Technologies**

Project no.: 045609

Bridge

International cooperation on Grid Technologies – IST Call 6

Specific target research project

Deliverable

D1.3 Prototype Implementation of Interoperable Grid Services

Start date of project: 1 January 2007

Duration: 24 months

Due date of deliverable: 31 September 2007

Actual submission date: 12 November 2007

Lead contractor for this deliverable: IT Innovation

Revision: 1.2

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination level		
PU	Public	✓
PP	Restricted to other programme participant (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract

This document describes the initial implementation of interoperable Grid services between GOSv2.1 and GRIA 5.1. In this report we provide an overview of our implementation of Phase 1 as described in Deliverable D1.2 “Report on Grid Infrastructure Interoperability Design” and go on to describe installation prerequisites and deployment instructions.

Revision history

Date	Version	Author	Modification
21.09.2007	0.1	E. Rowland Watkins	Initial Version
02.10.2007	0.2	E. Rowland Watkins Yongjian Wang	Fleshed out descriptions of Phase 1 components
04.10.2007	0.3	E. Rowland Watkins	Minor changes and added link to Apache 2 <code>gria-services.conf</code>
08.10.2007	0.4	Yongjian Wang E. Rowland Watkins	Updated descriptions to GOSv2.1 BES Job Service
18.10.2007	0.5	E. Rowland Watkins	Updates based on review and other developments
08.11.2007	1.0	Mike Boniface E. Rowland Watkins	Updates after internal review
12.11.2007	1.1	Yongjian Wang E. Rowland Watkins	Final contributions and corrections from BUAA
12.11.2007	1.2	Yongjian Wang E. Rowland Watkins	Minor corrections

Copyright

Copyright © University of Southampton IT Innovation Centre and Beihang University, 2007.

Table of contents

1	Introduction	4
1.1	Purpose	4
2	Phase 1 Implementation	4
2.1	Job Processing	5
2.2	Data Management	5
2.3	Security	6
2.3.1	Data Stager Security	6
2.4	Middleware Enhancements	7

2.4.1	GOSv2.1 Enhancements.....	7
2.4.2	GRIA 5.1 Enhancements.....	11
3	Installation and Deployment	14
3.1	Deployment	14
3.2	GOSv2.1 BES Job Service	15
3.2.1	Prerequisites	15
3.2.2	Additional Documentation	16
3.3	GRIA 5.1-bridge-1 Basic Applications Services	16
3.3.1	Prerequisites	16
3.3.2	Additional Documentation	16
4	Summary	16
5	Appendix A	17

1 Introduction

1.1 Purpose

This document is the deliverable D1.3 “*Prototype Implementation of Interoperable Grid Services*” of the EU IST-2006-045609 BRIDGE project, as specified in the Annex 1- “Description of Work” [1].

This deliverable describes the initial prototype implementation of interoperable Grid services between GOSv2.1 and GRIA 5.1. In this deliverable, we provide an overview of our implementation of Phase 1 as described in Deliverable D1.2 “*Report on Grid Infrastructure Interoperability Design*” and go on to describe installation prerequisites and deployment instructions.

2 Phase 1 Implementation

In deliverable D1.2 we described an incremental approach to interoperation between GOSv2.1 and GRIA 5.1, split into four phases. The intention of Phase 1 was to provide access to GOSv2.1 functionality but through a single interface.

Figure 1 depicts the implementation of Phase 1, where GOSv2.1 is completely encapsulated by GRIA 5.1. GRIA is able to communicate with GOS using an application wrapper script; this means GOSv2.1 is exposed to application workflow clients as a GRIA application.

¹ BRIDGE Annex 1 Description of Work, date of preparation 04/10/2006

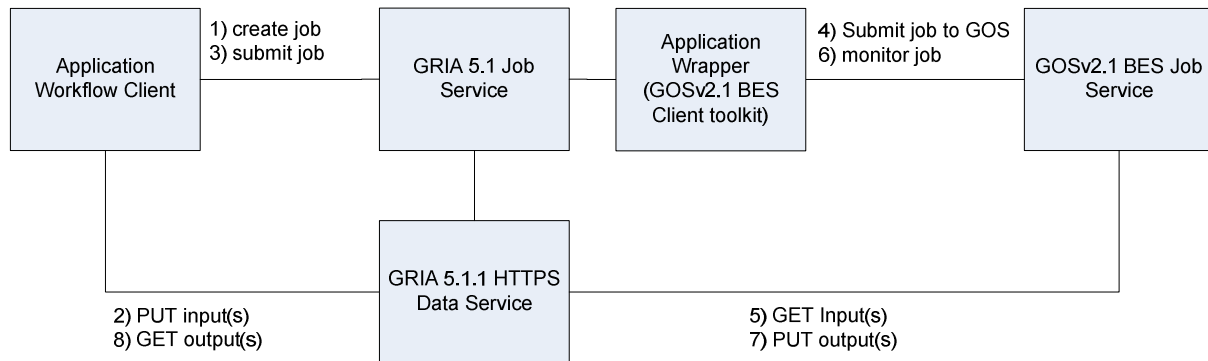


Figure 1. Phase 1 GOSv2.1 Encapsulation

When application workflow clients require jobs to be processed by GOSv2.1, they will use the GRIA application and submit jobs to that. The process is as follows:

1. Workflow client creates a job at GRIA Job Service
2. Workflow client uploads input data to GRIA Data Service (HTTP PUT)
3. Workflow client submits job to GRIA Job Service
4. GRIA application wrapper generates JSDL and submits new job to GOSv2.1 BES Job Service
5. GOSv2.1 BES Job Service downloads inputs from the GRIA Data Service (HTTP GET)
6. GRIA application wrapper monitors job at GOSv2.1 BES Job Service
7. GOSv2.1 BES Job Service uploads outputs to GRIA Data Service (HTTP PUT)
8. Workflow client downloads output from GRIA Data Service (HTTP GET)

2.1 Job Processing

As we showed in Figure 1, Phase 1 job processing is handled by the GRIA 5.1 Job Service. In cases where application workflow clients need applications provided by GOSv2.1, GRIA submits a new job to the GOSv2.1 BES Job Service and monitors until completion. Results from this job are then made accessible to the workflow client through the GRIA 5.1 Job Service.

2.2 Data Management

In deliverable D1.1, application workpackages specified the need for large throughput data transfer. While SOAP with Attachments is an option, it suffers with large payloads and when WS-Security is enabled for message security and HTTPS is used for transport layer security.

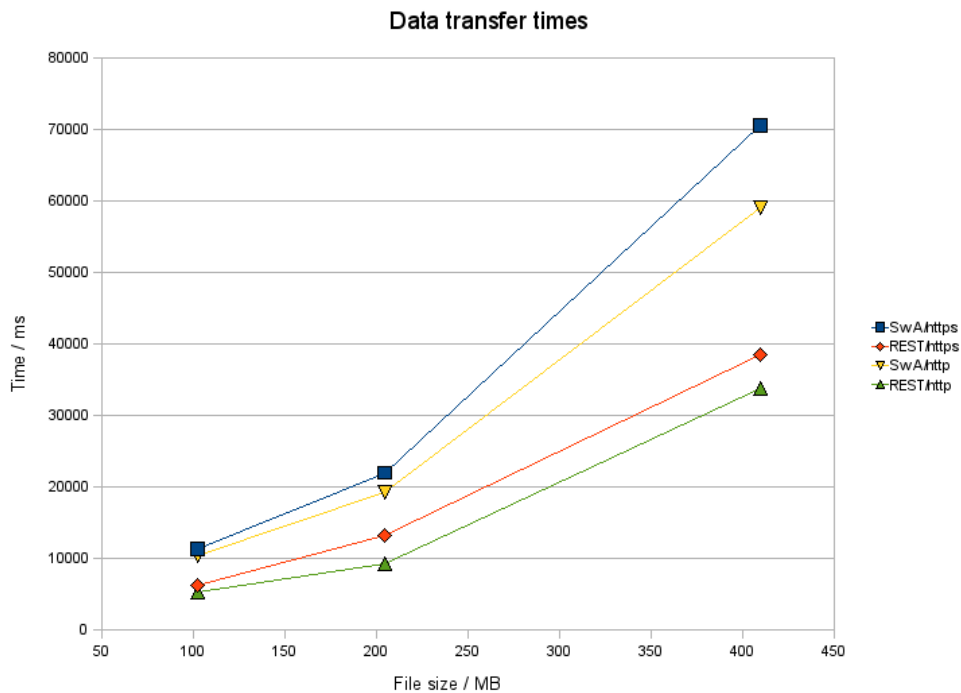


Figure 2. Data Transfers times for SwA and HTTP

Figure 2 shows some empirical results comparing the transfer times for 100, 200 and 400MB of data over SOAP with Attachments (HTTP and HTTPS) as well as ReST methods (Representational State Transfer – HTTP PUT and GET). As can be seen, PUT and GET using ReST is nearly twice as fast as SOAP with Attachments. It is highly likely that larger pay loads will have longer transfer times.

When using SOAP with Attachments, a large amount of processing must be performed to sign the data before the SOAP message is sent to its destination. HTTP data transfer, on the other hand, is capable of efficient data throughput even when Transport Layer Security is enabled. Unlike SOAP with Attachments which can only verify the data once transfer is complete, HTTPS is capable of verifying individual packets as they are received.

WP1 has extended the GRIA Data Service to support basic HTTP PUT and GET data transfer protocols. The use of standard HTTP means a greater selection of application tools can interoperate with the data services. HTTPS provides the necessary transport security (mutual authentication), where both the client and server authenticate each others' identity (see section 2.3)

2.3 Security

By default, GRIA 5.1 uses WS-Security for message level security and transport layer security. The GOSv2.1 BES job service is secured using transport layer security, but not message level security. During Phase 2, the GOSv2.1 BES job service will be fully integrated with GOS Agora service which will include message level security. The recommended deployment configuration for Phase 1 is to place GRIA in front of the GOSv2.1 BES job service in the same institution (see section 3.1).

2.3.1 Data Stager Security

One consequence of encapsulating the GOSv2.1 BES job service behind GRIA is that the service has access to all data stagers created by workflow clients; it is not possible to limit

access per job. This limitation is more pronounced since the GOSv2.1 BES job service is capable of running arbitrary shell scripts.

While this limitation exists in Phase 1, it will not in Phase 2. In Phase 2, the infrastructure will use the approach as describe in Deliverable D1.2, section 2.6.3. This approach requires a temporary key-pair to be created at the GOSv2.1 BES job service for each job. Access control of any data transfer between GRIA and GOS is therefore limited to a single job.

2.4 Middleware Enhancements

During initial development, both BUAA and IT Innovation have been working job processing and data management tasks to satisfy our Phase 1 design.

2.4.1 GOSv2.1 Enhancements

2.4.1.1 GOSv2.1 BES Job Service

OGSA-BES (Basic Execution Service) is a service to which clients can send requests to initiate, monitor, and manage computational activities. The specification defines an extensible *state model* for activities; an extensible *information model* for a BES and the activities that it creates; and three *port-types*; BES-Management, BES-Factory, and BES-Activity.

- BES-Management defines operations for managing the BES itself;
- BES-Factory defines operations for initiating, monitoring, and managing sets of activities, and for accessing information about the BES;
- BES-Activity defines operations for monitoring and managing individual activities.

At present, the GOSv2.1 BES Job Service implements BES-Factory port-types for job submission, monitoring and termination, and BUAA are working on the other two port-types. In this document, we give a brief introduction to the BUAA implementation.

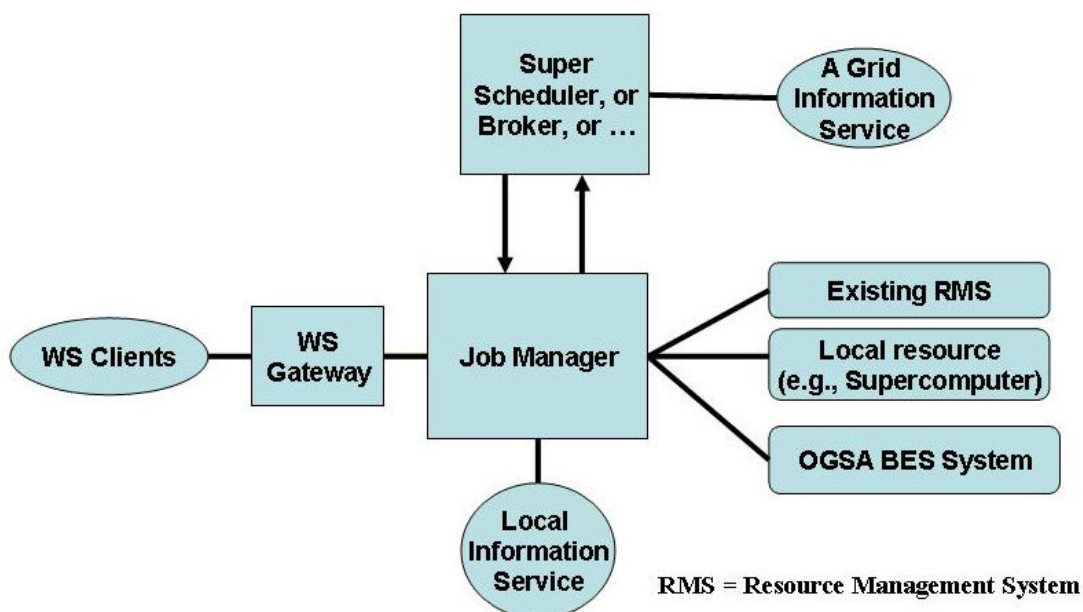


Figure 3. Conceptual Architecture of GOS Job Service

Figure 3 depicts the conceptual architecture of GOS Job Service similar to that presented in Deliverable D1.2 “Report on Grid Infrastructure Interoperability Design”; it consists of the following sub modules:

- WS Clients: Client API that end user can use to interact with Job Service;
- WS Gateway: Web Service façade;
- Job Manager: Maintains job state and invokes backend resource management system;
- Backend RMS: Different kinds of backend Resource management system now supported, including OpenPBS and LSF;
- Information Service: Maintains underlying resource information;
- Meta Scheduler: Chooses the most appropriate computing resource for batch job process.

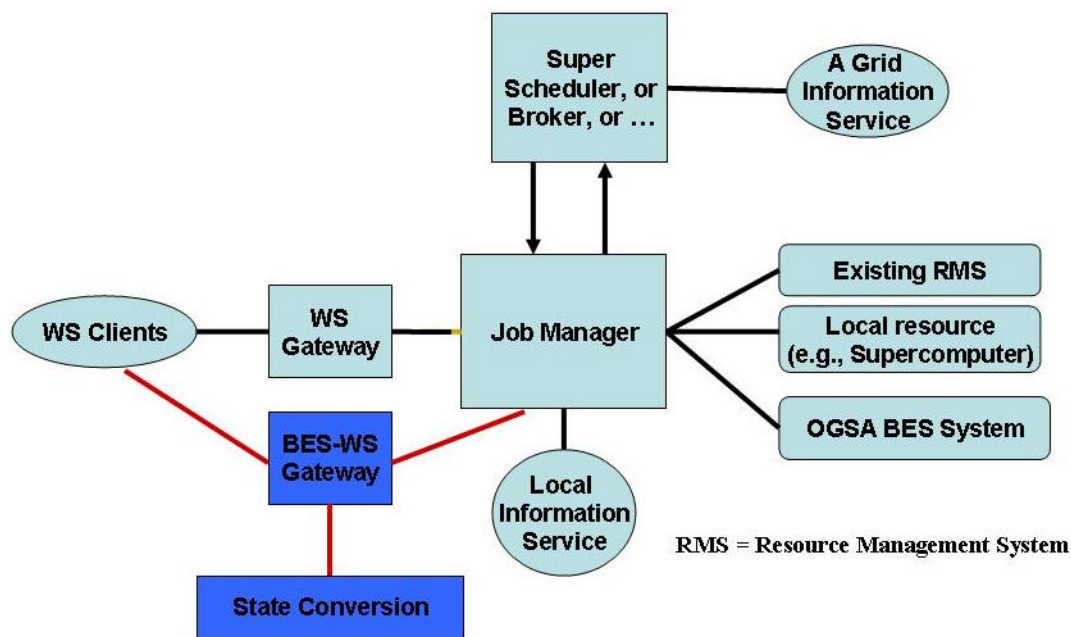


Figure 4. Conceptual Architecture of GOSv2.1 BES Service

Figure 4 depicts the conceptual architecture of GOSv2.1 BES Service. In addition to the existing components, BUAA have added two new components to support OGSA-BES:

- BES-WS-Gateway: Web Service façade, implements BES-Factory port-type based on GridSAM 2.0.1;
- State Conversion: OGSA-BES defines new state transition diagram, this component is used to do mapping between GOSv2.1 and OGSA-BES;

2.4.1.2 MPI type job Support in GOS Batch Service

MPI is very important in the HPC community. SIMM which is a Chinese partner in WP4 requires MPI for parallel job processing, which is not supported by GOS Batch Service by default. We provide a simple MPI support based on the fork job manager.

There are several difficulties in providing MPI support, which can be seen in the Dock6 application:

- Some MPI specific parameter need to set which is begin with #;
 - #PBS -l nodes=2:ppn=4 means this MPI job will be launched in 2 nodes, each node run 4 processes

- #PBS -q workq means job will be submitted to Queue named workq;
 - mpiexec ../../dock6/bin/dock6.mpi -i mpi.in -o mpi.out means the job should be executed by mpiexec command.
- When GOS processes the JSDL, the line beginning with # is considered as an annotation and will be omitted.

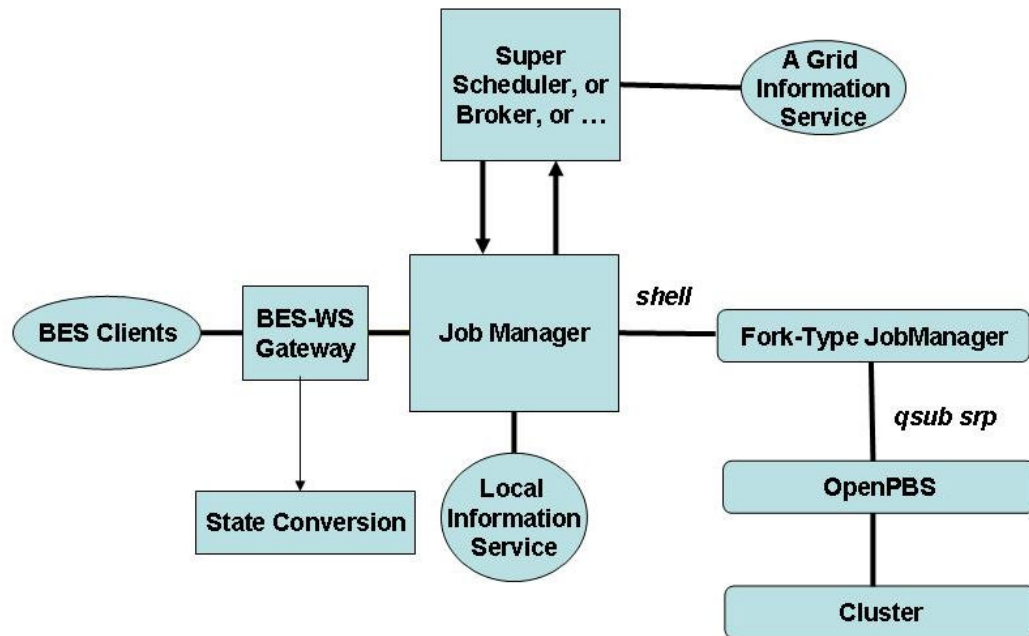


Figure 5. MPI Support in GOS Batch Service

MPI support in the GOS Batch service consists of two different parts:

- Fork-Type JobManager:
 - Fork JobManager is different from other RMS, it does not interact with remote resource manager, just launch a local process for job executing;
- Two shell scripts for executing MPI jobs:
 - Shell script executed by Fork-Type JobManager

```
#!/bin/sh
myFile=jobFinish
qsub job0.srp

while [ ! -f "$myFile" ];
do
    sleep 1
done
echo $myFile" exist. job finished."
tar -cvf output.tar output/
```

This script is executed by Fork-Type JobManager, it consists of three steps:

- Submit batch job to OpenPBS using qsub command
- Blocking until the submitted batch job finished
- Compress the files inside output folder using tar command
- Srp script which is submitted to OpenPBS

```
#This script file was automatically generated by Sunway Cluster
#UniqueID 100_wyj_1000_job0
#!/bin/sh
#PBS -S /bin/sh
#PBS -N ddggrid_dock6
#PBS -q workq
#PBS -l nodes=1:ppn=2
#PBS -j oe
source ~/.bash_profile
cd $PBS_O_WORKDIR
mkdir output
mpiexec ../../dock6/bin/dock6.mpi -i mpi.in -o mpi.out
touch jobFinish
```

This script is submitted to OpenPBS, it consists of the following steps:

- Define all the necessary parameter for MPI-Type job;
- Execute MPI-Type job using mpiexec command;
- Create jobFinish file to indicate the batch job has finished;

2.4.1.3 HTTP Data Transfer

GOS already supports six different transfer protocols: FTP, GSI-FTP, SFTP, FILE, HTTP and WebDAV, Apache-Commons VFS framework is used as integrating framework for underlying data transfer protocol. FTP is the default data transfer protocol. In the BRIDGE project, we have provided a new VFS protocol provider to support HTTP-based data transfer including GET, PUT.

Updates to GRIA (see section 2.4.2.2) enable the use of HTTPS to transfer data between different middleware. By default, however, the HTTP provider used by GOS does not support “PUT” method specified in HTTP protocol; part of Phase 1 development has been to add a new provider that can “PUT” data to HTTP server in GOS. In the interest of backwards compatibility and library reuse, rather than modifying the original HTTP provider, we have developed a new protocol scheme called “gria” that utilises HTTP as its transfer protocol to do data transfer between GOS and GRIA. We added this new provider to the VFS component of GOS, so a source or target URL in JSDL submitted to GOS now looks much like:

- gria://202.127.19.89:18080/gria-basic-app-services/data-stager/ff808081-160e874d-0116-0e9802f4-0001

2.4.1.4 GOSv2.1 BES client toolkit

GOSv2.1 BES Client Toolkit is the interface that enables the GRIA 5.1 Job Service and the GOSv2.1 BES Job Service to interoperate with one another.

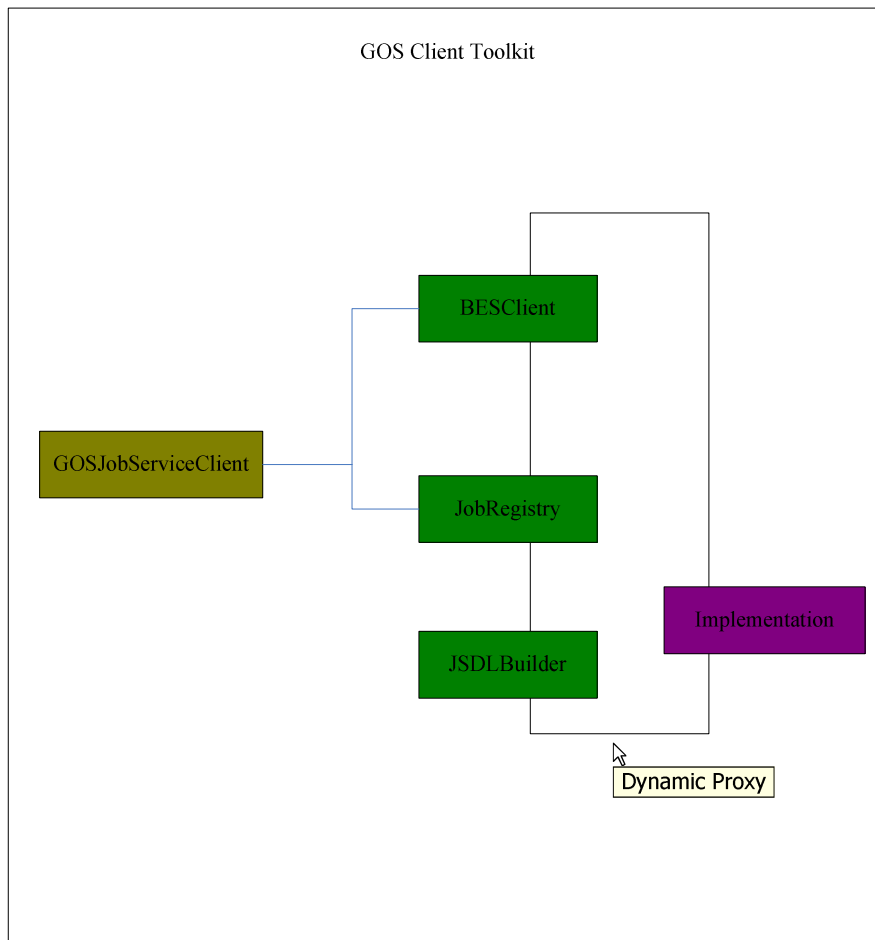


Figure 6. GOS BES client toolkit architecture

Figure 6 shows the architecture of the client toolkit. The BESClient class is used to encapsulate the client invocations to the GOSv2.1 BES Job Service. The JobRegistry class keeps information of current job, while the JSDLBuilder class generates JSDL documents submitted to GOSv2.1 BES Job Service. The interface to the BESClient class can be found in section 5.

2.4.2 GRIA 5.1 Enhancements

As part of the core development in Phase 1, IT Innovation has been working on enhancing GRIA’s Basic Application Services to support HTTPS data transfer that can support dynamic roots of trust, as well as application wrappers for encapsulating the GOSv2.1 BES Job Service.

2.4.2.1 Mutual Authentication with Dynamic Roots of Trust

The use of HTTPS with mutual authentication poses several issues regarding dynamic roots of trust. Typically, any HTTPS server requiring mutual authentication will only accept requests from clients with certificates signed by a well known root of trust (CA). In Apache 2 and Apache Tomcat, these well known roots of trust are stored in their respective configuration directories (“ca-bundle.crt” “cacerts.pem” respectively).

Configuring Apache Tomcat to support dynamic roots of trust is difficult, since Tomcat limits all roots of trust to the “<JAVA_JDK_INSTALL>/jvm/lib/security/cacerts.pem” file. To support client certificates from different, unknown roots of trust, Tomcat must have each CA

installed in the ““<JAVA_JDK_INSTALL>/jvm/lib/security/cacerts.pem” file, which would be a serious administrative issue.

Rather than use Apache Tomcat to handle HTTPS authentication, it is possible to configure Apache 2 to handle HTTPS requests, but use GRIA’s PBAC access control mechanism to support dynamic roots of trust. This means that Apache 2 does not verify the client certificate against its “ca-bundle.crt” file; PBAC effectively becomes part of the SSL negotiation before an HTTP request is honoured.

Figure 7 depicts the how PBAC is used in the SSL negotiation. Client HTTPS requests are handled by Apache 2, then passed (1) on to an HTTP servlet (2) that fronts the GRIA Data Service (4). The HTTP ReST interface to the data service works identically to that for SOAP messages, except that the X.509 certificate of the client is extracted from the SSL session rather than from the WSS4J results vector. There is no generic PBAC PEP for the ReST interface; the RestDataServlet class calls the PDP directly (3). There is no generic PBAC PEP for the ReST interface; the RestDataServlet class calls the PDP directly (3).

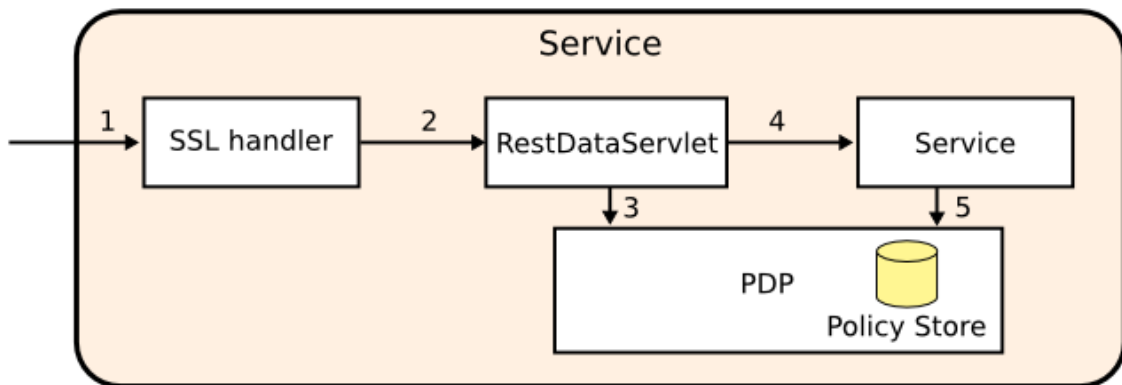


Figure 7. SSL with dynamic roots of trust

The reason for not using a generic PEP component is that the operation names correspond to the SOAP operation names (e.g. "read" and "save") rather than the ReST names (e.g. "GET" and "PUT"). By reusing the existing SOAP operation names, we are able to reuse existing PBAC policies rather than defining new ones.

2.4.2.2 HTTP Data Transfer

IT Innovation has enhanced the existing GRIA Data Service to support PUT and GET operations through HTTPS, with PUT operations secured using GRIA’s PBAC access control mechanism. Although data can now be accessed using HTTPS, access control rule updates are still done through the current SOAP interface.

The GRIA 5.1-bridge-1 Data Service supports the following HTTP methods: GET, PUT, DELETE.

2.4.2.3 GRIA-GOS Application Wrapper

Since GOSv2.1 is to be encapsulated by GRIA in Phase 1, IT Innovation has implemented a GRIA wrapper client that invokes the GOSv2.1 BES job service with the appropriate JSDL document. This wrapper can be seen in Figure 1 between GRIA and GOS. For this work, we have used the GOSv2.1BES client toolkit to generate the appropriate JSDL. The JSDL document includes both input and output data stagers, which point to the GRIA Data Service. An example JSDL document can be seen in below.

```

<JobDefinition xmlns="http://schemas.ggf.org/jSDL/2005/11/jSDL">
  <JobDescription>
    <JobIdentification>
      <JobName>Imagemagick job</JobName>
      <Description>Imagemagick description</Description>
      <JobAnnotation>no annotation</JobAnnotation>
      <JobProject>BRIDGE project</JobProject>
    </JobIdentification>
    <Application>
      <POSIXApplication
xmlns="http://schemas.ggf.org/jSDL/2005/11/jSDL-posix">
        <Executable>script.sh</Executable>
        <Argument> /source-image-1.jpg</Argument>
        <Argument> source-image-2.jpg</Argument>
        <Output>blended-image.jpg</Output>
        <Error>stderr.txt</Error>
      </POSIXApplication>
    </Application>
    <DataStaging>
      <FileName>source-image-1.jpg</FileName>
      <CreationFlag>overwrite</CreationFlag>
      <Source>
        <URI>gria://barkan.it-innovation.soton.ac.uk/gria-
basic-app-services/data-stager/502cbb8e-80e1-11dc-8314-
0800200c9a66</URI>
      </Source>
    </DataStaging>
    <DataStaging>
      <FileName>source-image-2.jpg</FileName>
      <CreationFlag>overwrite</CreationFlag>
      <Source>
        <URI>gria://barkan.it-innovation.soton.ac.uk/gria-
basic-app-services/data-stager/1297a13c-80e2-11dc-8314-
0800200c9a66</URI>
      </Source>
    </DataStaging>
    <DataStaging>
      <FileName>blended-image.jpg</FileName>
      <CreationFlag>overwrite</CreationFlag>
      <Target>
        <URI>gria://barkan.it-innovation.soton.ac.uk/gria-
basic-app-services/data-stager/
1df43a88-80e2-11dc-8314-
0800200c9a66</URI>
      </Target>
    </DataStaging>
  </JobDescription>
</JobDefinition>

```

In this example, we have an Imagemagick blend job (composite.exe) with three data stagers: two inputs and one output. In each case, the data stagers are accessible through HTTPS for inputs (HTTPS GET) and outputs (HTTPS PUT).

3 Installation and Deployment

In this section we provide basic information on the installation and deployment of Phase 1 components for application workpackage partners.

3.1 Deployment

For Phase 1 components to work effectively, each component must be installed and deployed in the correct manner along with any prerequisites software.

Figure 8 depicts a typical deployment of Phase 1 components. In the cases where GRIA is used, deployment is as recommended on the GRIA website. GOSv2.1 BES service deployments, however, should be hosted behind a GRIA installation that has the GRIA-GOS application wrapper script installed.

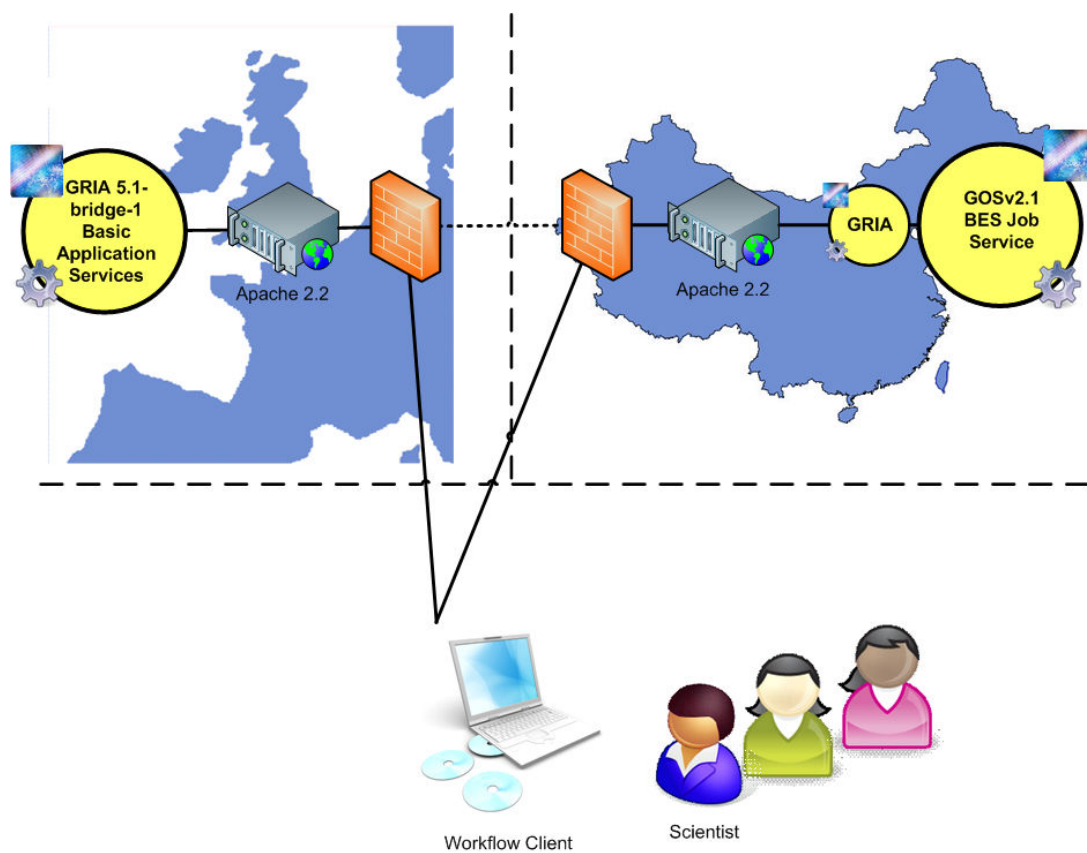


Figure 8. Deployment of Phase 1 components

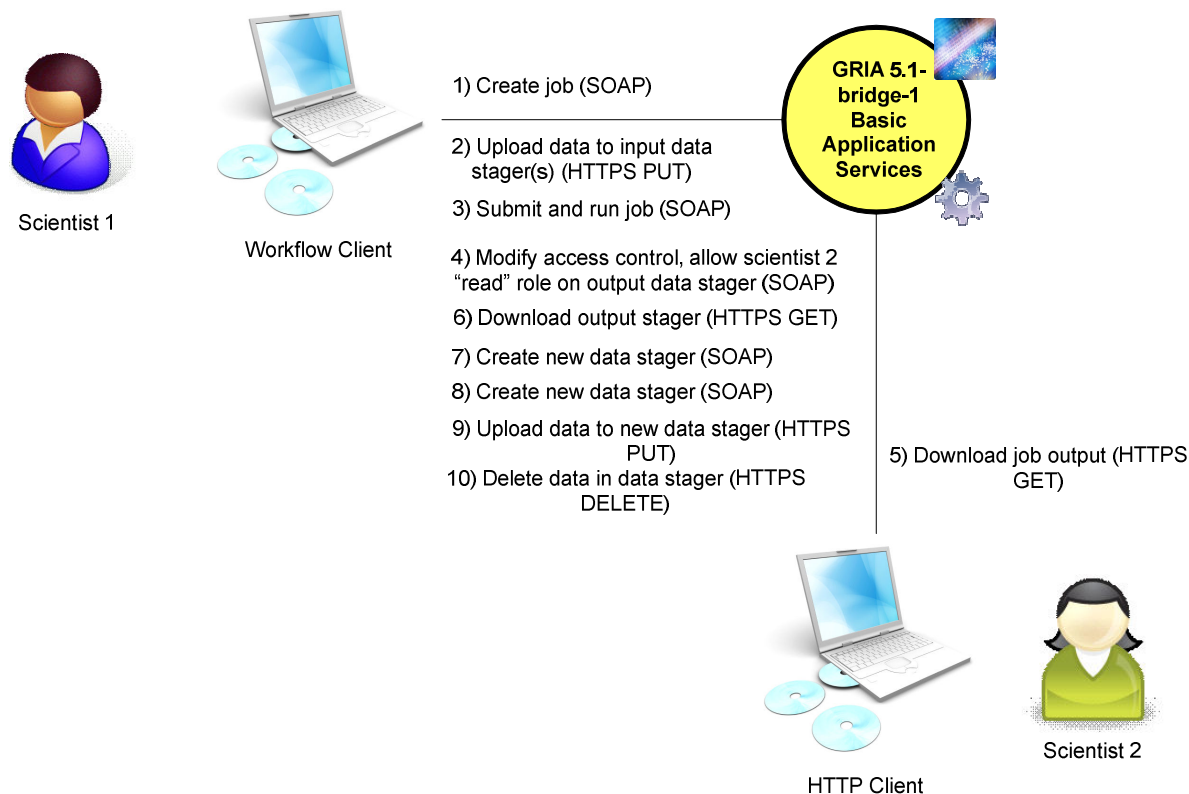


Figure 9. Example HTTPS data usage scenario

In Figure 9, the workflow user will go through a complete job lifecycle, creating an additional data stager to store job output. All data transfer and management can be done over HTTPS, whereas access control management remains under SOAP operations. The result of this is that workflow users are now able to use a common HTTP client (e.g. curl) to perform the upload with their X509 keypair:

```
curl --cert user-certificate.crt --key user-private-key.pem -k
-T file-to-upload\
```

<https://barkan.it-innovation.soton.ac.uk/gria-basic-app-services/data-stager/ff808181-152215bf-0115-221982b5-0002>

3.2 GOSv2.1 BES Job Service

3.2.1 Prerequisites

- Java 1.5
- Tomcat 5.x
- GOSv2.1 BES Job Service
- GOSv2.1 BES client toolkit
- Applications

3.2.2 Additional Documentation

Further documentation for configuring the GOSv2.1 BES job service can be found on the BSCW:

http://bscw.scai.fraunhofer.de/bscw/bscw.cgi/d124258/gos_bes_service_installation_instructions_1_0.doc

3.3 GRIA 5.1-bridge-1 Basic Applications Services

3.3.1 Prerequisites

- Java 1.5
- Tomcat 5.5.x
- GRIA 5.1-bridge-1 Basic Application Services
- Apache 2.2

3.3.2 Additional Documentation

The installation of the GRIA Data Service is no different from any other GRIA service package. The only complication is the setup of Apache 2.2, which forwards HTTPS connections to Tomcat. Further details on configuring transport layer security in Apache 2.2 (including mutual authentication) can be found here:

<http://www.gria.org/documentation/manual/service-installation-manual-all-services/configuring-transport-layer-security/apache>

When configuring the `gria-services.conf` it is important to uncomment the following line:

```
SSLVerifyClient optional_no_ca
```

This declaration instructs Apache 2 TLS to not check the CA of incoming certificates during TLS negotiation. As we described in section 2.4.2.1, checks against roots of trust are performed by PBAC.

A copy of the `gria-services.conf` can be found on the BSCW:

<http://bscw.scai.fraunhofer.de/bscw/bscw.cgi/d112996/gria-services-bridge.conf>

4 Summary

This deliverable described the components of the BRIDGE Phase 1 interoperation architecture implementation. The main components of Phase 1 include the GOSv2.1 BES Job Service, the GOSv2.1 BES client toolkit, and GRIA 5.1-bridge-1 Basic Application Services.

Work by BUAA has concentrated on producing a new GOSv2.1 BES Job Service and a client toolkit for partners to use. Work by IT Innovation has concentrated on using the GOSv2.1 BES client toolkit to encapsulate GOS for Phase 1 as well as producing an HTTPS (mutual authentication) interface to the GRIA Data Service. Documentation has been provided for all components for application workpackages to integrate with their respective applications.

5 Appendix A

```
public interface BESClient{
    org.w3c.dom.Document CreateActivity(org.w3c.dom.Document input);
    org.w3c.dom.Document GetActivityStatuses(org.w3c.dom.Document input);
    org.w3c.dom.Document TerminateActivities(org.w3c.dom.Document input) ;
    org.w3c.dom.Document GetActivityDocuments(org.w3c.dom.Document input) ;
}
public interface JobRegistry{
    String[] getJobIDs();
}
public interface JSDLBuilder {
    String getArgument();
    void setArgument(String argument);
    String getErrorFile();
    void setErrorFile(String errorFile);
    String getExecutable();
    void setExecutable(String executable);
    String getJobName();
    void setJobName(String jobName);
    String getOutputFile();
    void setOutputFile(String outputFile);
    String getUserName();
    void setUserName(String userName);
    boolean isValid();
    void setValid(boolean valid);
    void addStageIn(FileStageDesc desc);
    void removeStageIn(FileStageDesc desc);
    List getStageIns();
    void addStageOut(FileStageDesc desc);
    void removeStageOut(FileStageDesc desc);
    List getStageOuts();
    Document createJSDLForGridSAM();
    Document createJSDLForGOS();
}
public interface GOSJobServiceClient extends BESClient,JobRegistry{
}
```