Installing Globus Toolkit 4 and Programming Java Services

- A Laboratory Exercise Report -

Imre Matkó
mail: i07004@isi-hagenberg.at
web: www.risc.uni-linz.ac.at/home/imatko

Master's student at the
International School for Informatics
Johannes Kepler University and
University of Applied Sciences
Hagenberg im Mühlkreis,
Upper Austria, Austria

Hagenberg,
2008

Abstract

This report is the result of a laboratory work. It describes the basic steps of the installation and configuration of the Globus Toolkit 4 and the development and deployment of two sample services. Not all steps are described in detail but we indicate the corresponding parts from well known documentation of the topic. This work aims to present the steps of the work and to provide some further indications for tasks which were not obvious.

Table of Contents

1. Introduction

This report is the result of a laboratory work related to grid computing at ISI-Hagenberg. The tasks of the work were to install the Globus Toolkit 4 and provide a simple configuration and develop sample Java services. The work was guided by Professor Dr. Wolfgang Schreiner.

The steps described in this paper are based on the Globus administrator's guide [1], a handbook obout programming Java services for GT4 [2] and we have also used some material from the www.gt4book.com site.

In chapter 2 we give a step-by-step guide for preparing a machine and installing the Globus Toolkit as well as the steps required for a simple configuration.

Chapters 3 and 4 are describing the development of services and clients and finally in chapter 5 we explain the necessary steps for adding basic security features to our sample service. In the appendices we have all the relevant source codes for our services as well as a short description about their behavior.

In the paper we are using several acronyms and abbreviations, a list with their meanings can be found in the last chapter.

2. Software Installation

We need in our system to have certain pre-installed components, a list is given in the following chapter but we don't give installation and configuration instructions for these. The software installation can be performed easily following the step-by-step indications of the GT4 administrator's guide [1], we give an overview on this in chapter 2.2.

2.1. Prerequisites

A detailed list of the software prerequisites to install the Globus Toolkit can be found in the Admin Guide chapter 3.1. [1].

• J2SE 1.4.2+ SDK from Sun , IBM , HP , or BEA (do not use GCJ).

To install using Java 1.6 from a source installer, please apply the Java 1.6 patch7. Consult the admin guide for additional info.

• Ant 1.6+

Up to here it is enough for the WS Core setup.

• C compiler. If gcc, avoid version 3.2. 3.2.1 and 2.95.x are okay. gcc 4.1 has a bug that will trigger during the build of WS C (bug 431512). You can recompile the globus_js package from the advisories page, then run make again.

• C++ compiler. Use the version corresponding to your C compiler from the previous bullet.

• GNU tar

• GNU sed

• zlib 1.1.4+16

• GNU Make

• Perl 5.005 or newer. Some linux distributions may require additional perl library packages to be installed. Please see the prerequisites section specific to your linux distribution for details. In 4.0.5, XML::Parser is also required.

• sudo

• JDBC compliant database. For instance, PostgreSQL 7.1+, we used MySQL 5

• gpt-3.2autotools2004 (shipped with the installers, but required if building standalone GPT bundles/packages)

There are also some platform specific issues, please consult the admin guide to check whether your system falls in these.

2.2. Installation and Configuration

In this chapter we will describe in details only the steps which do not follow exactly the descriptions from the admin guide, can be easily omitted or can fail but are important.

Preliminary setup

System settings:

Create a home directory for the Globus Toolkit and set the GLOBUS_LOCATION environment variable to point to that.

Set up Java respectively the JAVA_HOME environment variable and update the PATH environment variable.

Set up apache ant and the ANT_HOME variable and eventually update the PATH, (e.g. /usr/local/globus-4.0.5.)

Create globus user, globus home dir and set owner/permissions.

Globus Installation

We used the Globus Toolkit debian binary install package. It can be downloaded from: http://www.globus.org/toolkit/downloads/4.0.6/ .

After unpacking the first step is to run *configure* and *make*.

Experience showed that it is possible sometimes that *make install* fails with a *gps not found* message. A solution (also available for other components, not just gps): manual build of gps and undo the previous make install's commands (it should suffice to delete all the installed files) and then run make install again.

Before continuing: some additional system settings are advised to make, however it is not indicated in the administrator's guide. For several tasks it is necessary to source the *globus-user-*

env.sh. It is practical to introduce the necessary command in an initialization file for all the user accounts which need it. If you are using Bourne shells this can be completed with the: *globus$ export GLOBUS_LOCATION=/path/to/install* command. In the case of csh the command is: *globus$ setenv GLOBUS_LOCATION /path/to/install* . It might be also handy to introduce in the system path the globus bin directory and maybe the etc directory also.

Simple Certificate Authority Setup

For our purposes now it is enough to use simple certificates. We have to generate simple self-signed certificates as indicated in chapter 6 and 7 of the admin guide [1].

At this point we will obtain a personalized hash (something like 1ab8433e), it is useful to note it, because we will have to use this later.

Now we should create the */etc/grid-security* folder and complete the GSI setup as indicated in the admin guide.

After we have created and signed user and host certificates we can perform the verification of the GSI setup with the following command: *user$ grid-proxy-init -debug -verify* .

Grid FTP service setup

The necessary steps are quite well described in chapter 8 of the admin guide. An important configuration step is to have in the /etc/services inserted the gridftp (gsiftp) and a corressponding inetd config entry, according to the admin guide, described at p.50.

Java WS Core setup

This component does not need any special configuration right now however on startup the container complains about RFT service... don't worry, RFT setup will follow. The **Java WS Core testing** should run with no errors. The necessary steps for testing can be found in the admin guide chapter 9.4. .

RFT setup

Normally there should be no problem, but it happened that the rft_schema_mysql.sql and rft_schema_mysql_pre4.0.sql were missing from the gt4.0.0 installation. I had to manually add them to the installation, in order to add mysql database support. They can be found in the installation directory.

After this can be downloaded, installed and set up mysql and mysql java connector described in detail in the admin guide at chapter 10.3.

These are the most important components for a Globus Toolkit installation if we want to develop and deploy simple services. For an integration in an existing Grid Environment we have to follow additional steps of setup and configuration but for now this suffices for our purposes.

3. Deployment of a Service

In this Laboratory Exercise we've programmed basic java web services for GT4. The toolkit gives an implementation of the WSRF specifications. Simple example files can be downloaded from the www.gt4book.com website.

The deployment of a web service can be done in five basic steps [2]:

a) Define a service interface with the WSDL language and realize the corresponding namespace mappings.

b) Implement the actual code of the service.

c) Define the deployment parameters with the WSDD language and provide a JNDI deployment file.

d) Compile and build a GAR file with the help of an Ant tool.

e) Deploy the service in GT4.

For the development we have followed the directory structure and used the scripts and configuration files provided in the examples from the www.gt4book.com site. Very simple hello world examples can be found in chapters 7.1 and 7.2 of the appendices.

a) We can put the WSDL interface file to any location. In our examples we have created corresponding subdirectories in the *schema* directory. In our case these files are the *Helloworld.wsdl* and the *HelloworldSecure.wsdl*, but the names can be arbitrary. In an interface file we have to specify all the parameters of our service which will realize the connection with the outer world. This file will consist of [2]: a **<definitions>** root element with the name and the namespace of the new service, the rest of it can be a verbatim copy from a sample; the **<portType>** element specifies the resource properties, operation names and the different messages realizing the communication with the outer world; the **<message>** element defines the input and output of each operation; the **<types>** element declares the request and response elements and specifies their types.

In our case for the hello world example we defined the HelloworldService name with the *http://local/examples/HelloworldService_instance* namespace having as input and output the "HelloInputMessage" and the "HelloOutputMessage" messages having the "hello" and "helloResponse" string type elements.

After this step we should specify some namespace mapping for the stub generator, which will generate java stubs out of the wsdl file. These mappings can be defined in the *namespace2package.mappings* file from the directory of the build tool. In our first example we introduced the following lines into this file (each mapping has to go in one line):

```
http\://local/examples/HelloworldService_instance=
local.examples.stubs.HelloworldService_instance

http\://local/examples/HelloworldService_instance/bindings=
local.examples.stubs.HelloworldService_instance.bindings

http\://local/examples/HelloworldService_instance/service=
local.examples.stubs.HelloworldService_instance.service
```

This is useful because by default we would get very long, automatically generated package names which are hard to handle.

b) The implementation consists of a short interface and the actual code of the service. The QName interface makes possible to use a qualified name when we will refer our service. An example can be found in the *HelloworldConstants.java* file from the *\local\examples\services\core\helloworld\impl\* folder. In the same folder is the service implementation, in the *HelloworldService.java* file. The folder hierarchy and the name of these files as well as their namespace specifications are important. They have to match each other and the overall target namespace of the service.

For the implementation we can follow the step-by-step indications from the Globus handbook [2], we just have to make sure to import the appropriate stubs with their names defined in the namespace mapping file described at point a) .

c) To make our service usable for the web container we have to write some deployment files. These are realized by the *deploy-jndi-config.xml* and the *deploy-server.wsdd* files, in our example contained in the *local\examples\services\core\helloworld\* folder. Both their names and the location are specified by standard. The first file is the wsdd deployment descriptor. It specifies for the container the URI, the class name, the wsdl file and some other options. The JNDI deployment files main role is to specify the resource home for our service. Both files have a simple and compact xml structure, their full code are included in the appendices.

d) With the help of the *globus-build-service.sh* script and the given *build.xml* Ant buildfile it is quite easy to create the gar file for our service. Before this it is recommended to edit the *build.mappings* file to make this task even easier. In this file we can specify some parameters which we would have to pass by command-line to the build script to locate the wsdl interface and the namespace of the service and map to these a single, unique name. For our helloworld service an entry in the mapping file looks like as follows:

```
helloworld, local/examples/services/core/helloworld,
schema/local/examples/HelloworldService_instance/Helloworld.wsdl
```

Now we can invoke the build script with the parameter *helloworld* : *./globus-build-service.sh helloworld*

We note that the `./globus-build-service.py` is the python version which can be used under Windows with the same parameters.

If we do not make the above mentioned mapping this will look like:

```
./globus-build-service.sh \

   -d  local/examples/services/core/helloworld \

   -s schema/local/examples/HelloworldService_instance/Helloworld.wsdl
```

e) If the previous step was successful now we should have some gar file with the full name of our service: *local_examples_services_core_helloworld.gar*. This archive can be deployed into the web service container with the Globus command: *globus-deploy-gar local_examples_services_core_helloworld.gar.* To undeploy it we can use the *globus-undeploy-gar local_examples_services_core_helloworld.gar* command.

4. Usage of Services

Once we have our services deployed in the Globus container it is possible to invoke their operations from any machine on the network or the Internet, and to retrieve the corresponding results. This needs a client application, what is completely independent from the implementation of the service. Thus it suffices to know just the interface and the corresponding stubs.

Our sample client class is called *Client.java*. The source code is included in the appendices and among the sample files it is placed in the *local\examples\clients\HelloworldService_instance\* directory.

The client needs to import all the stubs related to the operations of the target service and the addressing locator stub. In the actual code we shall create a so called Endpoint Reference Type endpoint (`EndpointReferenceType endpoint = new EndpointReferenceType();`) and set the services URI (`endpoint.setAddress(new Address(serviceURI));`). This will be used to connect to the service. The reference to the service is obtained via the addressing locator stub, it returns in our case a *HelloworldPortType* object (`HelloworldPortType Helloworld = locator.getHelloworldPortTypePort(endpoint);`). This can be used locally however we have to be aware that remote exceptions might be thrown so try-catch blocks are needed.

Before we try to run or compile the code we have import the Globus development environment, this can be done with the command: *source $GLOBUS_LOCATION/bin/etc/globus-devel-env.sh* for bash or *source $GLOBUS_LOCATION/bin/etc/globus-devel-env.csh* for CSH sehlls.

Both for compiling and running the client we need to have some extra directories in the classpath:

```
javac -classpath ./build/stubs/classes/:$CLASSPATH \
local/examples/clients/HelloworldService_instance/Client.java
java -classpath ./build/stubs/classes/:.:$CLASSPATH \
local.examples.clients.HelloworldService_instance.Client \
http://127.0.1.1:8080/wsrf/services/local/examples/core/helloworld/HelloworldService
```

Notice that the Globus container should be started without security, with the *globus-start-container -nosec* command, because we have not set up the transport level security. For further explanations you can consult chapters 6.6 and Part III of the Globus Toolkit Service Programming handbook [2].

5. Secured Services

So far our hello world example was running without any form of security, it could be reached, invoked by any user or client having the corresponding stubs. The second example, the secure hello world sample service, implements basic security. In this case the service will require the user running the client to have valid digital certificates.

Both the service and the client implementation remains essentially the same, we will describe here the differences between our two examples. Details about the required steps can be found in chapter 17 of the GT4 Programming handbook [2].

First we have to provide a new, security descriptor file for our implementation. The name of the file in our example is *security-config-first.xml* and it is placed in the folder *local\examples\services\core\helloworldsecure\etc\* .

The most simple security descriptor looks like as follows:

```
<securityConfig xmlns="http://www.globus.org">

 <authz value="none"/>

</securityConfig>
```

This specifies that we do not require any authorization, the client is free to chose any type of security. For the service to be aware of this configuration we have to add a security descriptor element to the service deployment wsdd file (*deploy-server.wsdd*):

```
<parameter name="securityDescriptor"
value="etc/local_examples_services_core_helloworldsecure/security-config-
first.xml"/>
    </service>
```

The implementation has to be slightly changed, we have to add some new imports:

```
import org.globus.gsi.jaas.JaasSubject;
import org.globus.wsrf.NoSuchResourceException;
import org.globus.wsrf.ResourceContext;
import org.globus.wsrf.ResourceContextException;
import org.globus.wsrf.security.SecurityManager;
```

In our example we have also added some logging to demonstrate the usage of the security manager. The logging is done via the log4j component. In order to enable the logging for our class we have to edit the log4j properties file (*container-log4j.properties*) and add our class and the corresponding log level.

In our client we chose to use GSI Secure Conversation with encryption. This requires some additional classes to import compared to a non-secured version:

```
import javax.xml.rpc.Stub;
import org.globus.axis.util.Util;
import org.globus.wsrf.impl.security.authorization.NoAuthorization;
import org.globus.wsrf.security.Constants;
import org.oasis.wsrf.properties.GetResourcePropertyResponse;
```

In the code we need two further steps to initialize security:

```
((Stub)Helloworld)._setProperty(Constants.GSI_SEC_CONV,
Constants.ENCRYPTION);

((Stub)Helloworld)._setProperty(Constants.AUTHORIZATION,
NoAuthorization.getInstance());
```

The first command chooses the security type and the second one specifies client side authorization. It is also important to introduce these commands in extra try-catch blocks because they can throw various exceptions if any error occurs.

We note that the Globus container still has to be started without transportation level security (with the -nosec option).

The full source codes of can be found in the second part of the appendices.

6. References

[1] **GT4 Admin Guide**, November 2005 - http://www.globus.org/toolkit/docs/4.0/admin/docbook/

[2] Sotomayor, B. & Childers, L. (2006), **Globus Toolkit 4 – Programming Java Services**, Elsevier Inc., Morgan Kaufmann Publishers, San Francisco, USA

7. Appendices

Here you can find the full source codes and afferent configuration files of our two sample services. The first is a hello world service without any type of security while the second one is an adaptation of the first one with simple security.

1 Hello World Service

The hello world service is a simple Globus Java Service which can perform a single operation (hello). This operation takes one string argument as input and returns a string response containing the input value and some extra text. The service requires no security thus it can be invoked without any restrictions. The client takes an optional argument from the command line and passes it to the service or invokes it with some default parameter if the argument list is empty. Here we provide all the relevant sources:

WSDL Description File (*Helloworld.wsdl*)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloworldService"
 targetNamespace="http://local/examples/HelloworldService_instance"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:tns="http://local/examples/HelloworldService_instance"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">


<!--=============================================================

 T Y P E S

 =============================================================-->
<types>
<xsd:schema
targetNamespace="http://local/examples/HelloworldService_instance"
 xmlns:tns="http://local/examples/HelloworldService_instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">


 <!-- REQUESTS AND RESPONSES -->

 <xsd:element name="hello" type="xsd:string"/>
 <xsd:element name="helloResponse" type="xsd:string"/>

 <!-- RESOURCE PROPERTIES -->

 <xsd:element name="Value" type="xsd:string"/>

 <xsd:element name="HelloworldResourceProperties">
 <xsd:complexType>
 <xsd:sequence>
 <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
 </xsd:sequence>
 </xsd:complexType>
 </xsd:element>

</xsd:schema>
</types>


<!--============================================================

 M E S S A G E S

 ============================================================-->
<message name="HelloInputMessage">
 <part name="parameters" element="tns:hello"/>
</message>
<message name="HelloOutputMessage">
                                            <part name="parameters"
element="tns:helloResponse"/>
</message>

<!--============================================================

                       P O R T T Y P E

  ============================================================-->
<portType name="HelloworldPortType"
    wsrp:ResourceProperties="tns:HelloworldResourceProperties">

                                            <operation name="hello">
                                               <input
message="tns:HelloInputMessage"/>
                                               <output
message="tns:HelloOutputMessage"/>
                                            </operation>

</portType>

</definitions>
```

WSDD Deployment Descriptor (***deploy-server.wsdd***)

```
     <?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="local/examples/core/helloworld/HelloworldService"
provider="Handler" use="literal" style="document">
        <parameter name="className"
value="local.examples.services.core.helloworld.impl.HelloworldService"/>
        <wsdlFile>share/schema/local/examples/HelloworldService_instance/H
elloworld_service.wsdl</wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>

</deployment>
```

JNDI Deployment File (*deploy-jndi-config.xml*)

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">

<service name="local/examples/core/helloworld/HelloworldService">
      <resource name="home"
type="org.globus.wsrf.impl.ServiceResourceHome">
      <resourceParams>

            <parameter>
                  <name>factory</name>
                  <value>org.globus.wsrf.jndi.BeanFactory</value>
            </parameter>

      </resourceParams>

      </resource>
</service>

</jndiConfig>
```

Service QNames Interface (*HelloworldConstants.java*)

```
package local.examples.services.core.helloworld.impl;

import javax.xml.namespace.QName;

public interface HelloworldConstants {
                                                   public static final
String NS = "http://local/examples/HelloworldService_instance";

                                                   public static final
QName RP_VALUE = new QName(NS, "Value");

                                                   public static final
QName RESOURCE_PROPERTIES = new QName(NS,
                                                            "HelloworldReso
urceProperties");
}
```

The Service Implementation (*HelloworldService.java*)

```
package local.examples.services.core.helloworld.impl;

import java.rmi.RemoteException;

//import local.examples.stubs.HelloworldService_instance.HelloResponse;

import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.impl.ReflectionResourceProperty;
import org.globus.wsrf.impl.SimpleResourcePropertySet;

public class HelloworldService implements Resource, ResourceProperties {

                                                  /* Resource Property set
*/
                                                  private
ResourcePropertySet propSet;
                                                  /* Required by interface
ResourceProperties */
                                                  public
ResourcePropertySet getResourcePropertySet(){
                                                  return this.propSet;
                                                  }

                                                  /* Resource properties
*/
                                                  private String value;

                                                  /* Constructor.
Initializes RPs */
                                                  public
HelloworldService() throws RemoteException {
                                                     /* Create RP set */
                                                     this.propSet = new
SimpleResourcePropertySet(


HelloworldConstants.RESOURCE_PROPERTIES);

                                                     /* Initialize the
RP's */
                                                     try {

                                                  ResourceProperty valueRP
= new ReflectionResourceProperty(


HelloworldConstants.RP_VALUE, "Value", this);


this.propSet.add(valueRP);

                                                  setValue("Hello! - ");

                                                     } catch (Exception e)
{
                                                           throw new
RemoteException(e.getMessage(), e);
                                                     }
                                                  }

                                                  /* Get/Setters for the
RPs */
                                                  public String getValue()
{
                                                     return value;
                                                  }

                                                  public synchronized void
setValue(String value) {
                                                     this.value = value;
                                                  }


                                                  /* Remotely-accessible
operations */

                                                  public synchronized
String hello(String s) throws RemoteException {
                                                     value = "Service
received: " + s + "; ";
                                                     return value;
                                                  }


}
```

The Client Implementation (*Client.java*)

```
package local.examples.clients.HelloworldService_instance;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;

import local.examples.services.core.helloworld.impl.HelloworldConstants;
//import local.examples.stubs.HelloworldService_instance.HelloResponse;
import local.examples.stubs.HelloworldService_instance.HelloworldPortType;
import
local.examples.stubs.HelloworldService_instance.service.HelloworldServiceA
ddressingLocator;

//import javax.xml.rpc.Stub;

/*import org.globus.axis.util.Util;
import org.globus.wsrf.impl.security.authorization.NoAuthorization;
import org.globus.wsrf.security.Constants;
import org.oasis.wsrf.properties.GetResourcePropertyResponse;
*/

public class Client {

                                                  public static void
main(String[] args) {


HelloworldServiceAddressingLocator locator = new
HelloworldServiceAddressingLocator();

                                                     try {
                                                           String
serviceURI = args[0];


                                                           // Create
endpoint reference to service


                                                  EndpointReferenceType
endpoint = new EndpointReferenceType();
```

```
                                                                 endpoint.setAddress(new
Address(serviceURI));

                                                                          // Get PortType

                                                                 HelloworldPortType
Helloworld = locator.getHelloworldPortTypePort(endpoint);

                                                                          String param =
"Client";
                                                                          try{
                                                                              param =
args[1];
                                                                          }
                                                                          catch
(ArrayIndexOutOfBoundsException e){
                                                                          }

                                                                          // Perform an
operation

System.out.println("Client> service response value: "
                                                                                      +
Helloworld.hello(param));

                                                                    } catch (Exception e)
{

                                                                 e.printStackTrace();
                                                                    }
                                                                 }

}
```

2 *Secure Hello World Service*

The secure hello world service is a Globus Java Service with simple security, extending the service described in the previous part. The service can perform a single operation (hello). This operation takes one string argument as input and returns a string response containing the input value and some extra text. The client is free to choose any type of security however it is required for the client user to have a valid digital certificate for the Grid. The client takes an optional argument from the command line and passes it to the service or invokes it with some default parameter if the argument list is empty. Here we provide all the relevant sources:

The Security Descriptor (*security-config-first.xml*)

```
<securityConfig xmlns="http://www.globus.org">

 <authz value="none"/>

</securityConfig>
```

WSDL Description File (*Helloworld.wsdl*)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloworldSecureService"
    targetNamespace="http://local/examples/HelloworldSecureService_instanc
e"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://local/examples/HelloworldSecureService_instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


<!--============================================================

                      T Y P E S

  ============================================================-->
<types>
<xsd:schema
targetNamespace="http://local/examples/HelloworldSecureService_instance"
    xmlns:tns="http://local/examples/HelloworldSecureService_instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


      <!-- REQUESTS AND RESPONSES -->

      <xsd:element name="hello" type="xsd:string"/>
      <xsd:element name="helloResponse" type="xsd:string"/>
          <!--xsd:complexType/-->
      <!--/xsd:element-->

      <!-- RESOURCE PROPERTIES -->

      <xsd:element name="Value" type="xsd:string"/>

      <xsd:element name="HelloworldSecureResourceProperties">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="tns:Value" minOccurs="1"
maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>
      </xsd:element>

</xsd:schema>
</types>


<!--============================================================

                       M E S S A G E S

  ============================================================-->
<message name="HelloInputMessage">
      <part name="parameters" element="tns:hello"/>
</message>
<message name="HelloOutputMessage">
      <part name="parameters" element="tns:helloResponse"/>
</message>

<!--============================================================
                        P O R T T Y P E

  ============================================================-->
<portType name="HelloworldSecurePortType"
    wsrp:ResourceProperties="tns:HelloworldSecureResourceProperties">

      <operation name="hello">
            <input message="tns:HelloInputMessage"/>
            <output message="tns:HelloOutputMessage"/>
      </operation>

</portType>

</definitions>
```

WSDD Deployment Descriptor (*deploy-server.wsdd*)

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service
name="local/examples/core/helloworldsecure/HelloworldSecureService"
provider="Handler" use="literal" style="document">
        <parameter name="className"
value="local.examples.services.core.helloworldsecure.impl.HelloworldSecure
Service"/>
        <wsdlFile>share/schema/local/examples/HelloworldSecureService_inst
ance/HelloworldSecure_service.wsdl</wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass"
value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="loadOnStartup" value="true"/>

      <!-- SECURITY-->
      <parameter name="securityDescriptor"
value="etc/local_examples_services_core_helloworldsecure/security-config-
first.xml"/>
    </service>

</deployment>
```

JNDI Deployment File (*deploy-jndi-config.xml*)

```
<?xml version="1.0" encoding="UTF-8"?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">

<service
name="local/examples/core/helloworldsecure/HelloworldSecureService">
                                                 <resource name="home"
type="org.globus.wsrf.impl.ServiceResourceHome">
                                                 <resourceParams>

                                                    <parameter>
                                                    <name>factory</name>

<value>org.globus.wsrf.jndi.BeanFactory</value>
                                                       </parameter>

                                                    </resourceParams>

                                                    </resource>
</service>

</jndiConfig>
```

Service QNames Interface (*HelloworldConstants.java*)

```
package local.examples.services.core.helloworldsecure.impl;

import javax.xml.namespace.QName;

public interface HelloworldSecureConstants {
                                              public static final String NS
= "http://local/examples/HelloworldSecureService_instance";

                                              public static final QName
RP_VALUE = new QName(NS, "Value");

                                              public static final QName
RESOURCE_PROPERTIES = new QName(NS,
                                                     "HelloworldSecureReso
urceProperties");
}
```

The Service Implementation (*HelloworldService.java*)

```
package local.examples.services.core.helloworldsecure.impl;

import java.rmi.RemoteException;

//import
local.examples.stubs.HelloworldSecureService_instance.HelloResponse;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.impl.ReflectionResourceProperty;
import org.globus.wsrf.impl.SimpleResourcePropertySet;

import org.globus.gsi.jaas.JaasSubject;
import org.globus.wsrf.NoSuchResourceException;
import org.globus.wsrf.ResourceContext;
import org.globus.wsrf.ResourceContextException;
import org.globus.wsrf.security.SecurityManager;
```

```
public class HelloworldSecureService implements Resource,
ResourceProperties {

                                             static Log logger =
LogFactory.getLog(HelloworldSecureService.class);

                                             /* Resource Property set */
                                             private ResourcePropertySet
propSet;
                                             /* Required by interface
ResourceProperties */
                                             public ResourcePropertySet
getResourcePropertySet(){
                                             return this.propSet;
                                             }

                                             /* Resource properties */
                                             private String value;

                                             /* Constructor. Initializes
RPs */
                                             public
HelloworldSecureService() throws RemoteException {
                                                /* Create RP set */
                                                this.propSet = new
SimpleResourcePropertySet(

HelloworldSecureConstants.RESOURCE_PROPERTIES);

                                                /* Initialize the RP's */
                                                try {
                                                      ResourceProperty
valueRP = new ReflectionResourceProperty(

HelloworldSecureConstants.RP_VALUE, "Value", this);

                                             this.propSet.add(valueRP);
                                                      setValue("Hello! -
");

logger.info("HELLOWORLDSERVICE: Mission Control, this is Jupiter I, the
Robinsons are all tucked in, we are ready to fly!\n");

                                                } catch (Exception e) {
                                                      throw new
RemoteException(e.getMessage(), e);
                                                }
                                             }

                                             /* Get/Setters for the RPs */
                                             public String getValue() {
                                                return value;
                                             }

                                             public synchronized void
setValue(String value) {
```

```
                                            this.value = value;
                                         }

                                         /* Remotely-accessible
operations */

                                         public synchronized String
hello(String s) throws RemoteException {
                                            logSecurityInfo("hello");
                                            value = "Service received:
" + s + "; ";
                                            return value;
                                         }

                                         private void
logSecurityInfo(String mN) //param - callers ID/name
                                         {
                                             logger.info("SECURITY INFO
FOR METHOD '" + mN + "'");


                                             //Print out the caller
                                             String id =
SecurityManager.getManager().getCaller();
                                             logger.info("The caller
is:" + id);
                                         }

}
```

The Client Implementation (*Client.java*)

```
package local.examples.clients.HelloworldSecureService_instance;

import
local.examples.services.core.helloworldsecure.impl.HelloworldSecureConstan
ts;
//import
local.examples.stubs.HelloworldSecureService_instance.HelloResponse;
import
local.examples.stubs.HelloworldSecureService_instance.HelloworldSecurePort
Type;
import
local.examples.stubs.HelloworldSecureService_instance.service.HelloworldSe
cureServiceAddressingLocator;

import javax.xml.rpc.Stub;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;
import org.globus.axis.util.Util;
import org.globus.wsrf.impl.security.authorization.NoAuthorization;
import org.globus.wsrf.security.Constants;
import org.oasis.wsrf.properties.GetResourcePropertyResponse;


public class Client {

                                         public static void
main(String[] args) {
```

```
HelloworldSecureServiceAddressingLocator locator = new
HelloworldSecureServiceAddressingLocator();

                                                    try {
                                                         String serviceURI =
args[0];

                                                         // Create endpoint
reference to service
                                                         EndpointReferenceType
endpoint = new EndpointReferenceType();

                                             endpoint.setAddress(new
Address(serviceURI));

                                                         // Get PortType

                                             HelloworldSecurePortType
Helloworld = locator.getHelloworldSecurePortTypePort(endpoint);

                                                         String param =
"Client";
                                                         try{
                                                             param = args[1];
                                                         }
                                                         catch
(ArrayIndexOutOfBoundsException e){
                                                         }

                                                         //Setup security


((Stub)Helloworld)._setProperty(Constants.GSI_SEC_CONV,
Constants.ENCRYPTION);


((Stub)Helloworld)._setProperty(Constants.AUTHORIZATION,
NoAuthorization.getInstance());

                                                         // Perform an
operation
                                                         try{

                                             System.out.println("Client>
service response value: "
                                                                     +
Helloworld.hello(param));
                                                         }
                                                         catch(Exception e){

System.out.println("[hello] ERROR: " + e.getMessage());
                                                         }

                                                    } catch (Exception e) {
                                                         e.printStackTrace();
                                                    }
                                             }

}
```

8. Abbreviations, Acronyms

GAR	Grid ARchive
GSI	Grid Security Infrastructure
GT4	Globus Toolkit 4
Java WS Core	Java Web Services Core, implements te WSRF and the WSN standards
JNDI	Java Naming and Directory Interface
RFT	Reliable File Transfer
URI	Universal Resource Identifier
VO	Virtual Organisation
WSDD	Web Service Deployment Descriptor
WSDL	Web Service Description Language
WSN	Web Service Notification
WSRF	Web Services Resource Framework