

5-2004

Classroom Exercises for Grid Services

Amy Apon

Clemson University, aaon@clemson.edu

Jens Mache

Yuriko Yara

Kurt Landrus

Follow this and additional works at: https://tigerprints.clemson.edu/computing_pres



Part of the [Computer Sciences Commons](#)

Recommended Citation

Apon, Amy; Mache, Jens; Yara, Yuriko; and Landrus, Kurt, "Classroom Exercises for Grid Services" (2004). *Presentations*. 5.
https://tigerprints.clemson.edu/computing_pres/5

This is brought to you for free and open access by the School of Computing at TigerPrints. It has been accepted for inclusion in Presentations by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

Classroom Exercises for Grid Services

Amy Apon, Jens Mache^{L&C}
Yuriko Yara, Kurt Landrus



Grid Computing

- Grid computing is way of organizing computing resources so that they can be flexibly and dynamically allocated and accessed
- There is a lot of research-related material on Grid computing, but very little that focuses on Grid computing in an undergraduate classroom setting

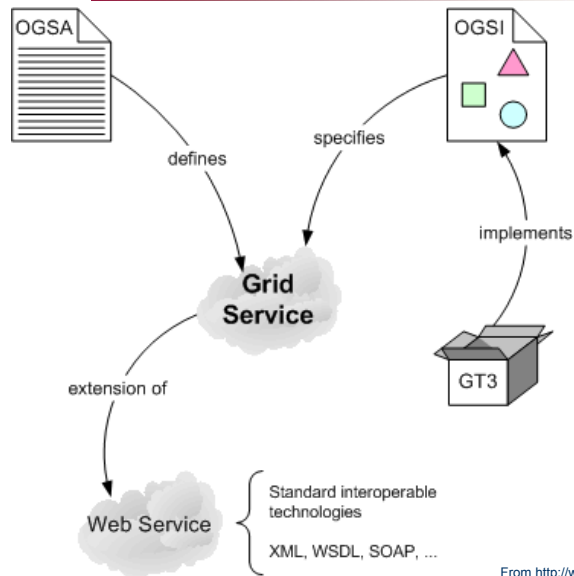


Overall Goals of this work

- Remove some of the mystery of Grid computing for students (alphabet soup!)
 - OGSA – Open Grid Services Architecture is a standard developed by the Global Grid Forum (GGF)
 - OGSI – Open Grid Services Infrastructure
 - GT3 – Globus Toolkit version 3
- Describe the core knowledge units for Grid
- Create and evaluate a set of exercises that can be used in a classroom setting in conjunction with existing literature



Amy Apon, Ph.D. • University of Arkansas • May 20, 2004 • Page 3



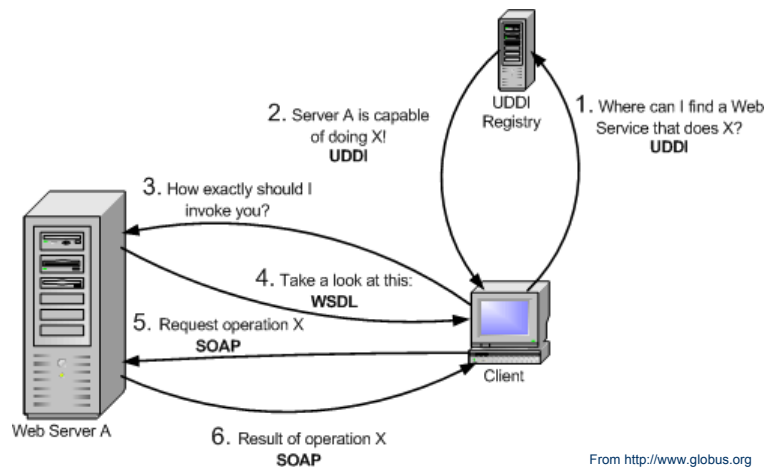
Amy Apon, Ph.D. • University of Arkansas • May 20, 2004 • Page 4

Grid Services

- Common interface specification supports the interoperability of discrete, independently developed **services**
- Concept similar to Remote Procedure Call (RPC), Remote Method Invocation (RMI), only applied over HTTP
- Based on extensions of Web Services



Web Services

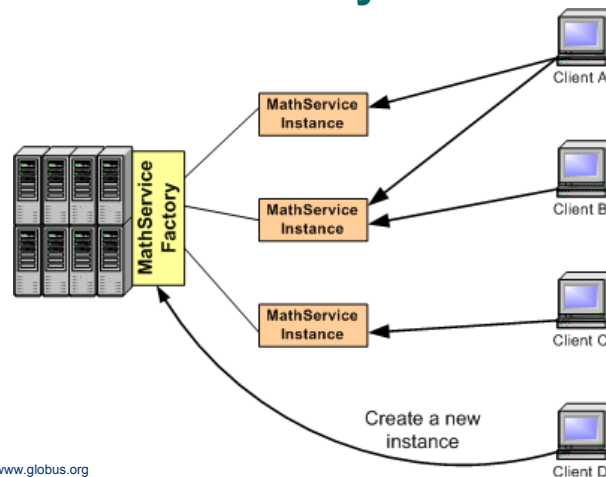


Web Services missing features

- At the time the OGIS V1.0 spec was published there was a gap between the need to define stateful Web Services and what was provided by the latest version of Web Services in WSDL 1.1 – Web Services were stateless and non-transient
- The result was the definition in OGIS of *Service Data* – a common mechanism to expose a service instance's state data for query, update, and change notification
- Also, Grid Services uses a *Factory* to manage instances – to allow transient and private instances



Grid Services Factory



From <http://www.globus.org>



Web Services Exercise

Motivation: Most undergraduates are familiar with user-level web access and client/server systems, but not the concept of remote procedure call or data marshalling – need 2-3 lectures on these topics before the exercise

- Marshalling using XML
- HTTP, SOAP, WSDL
- Semantics: at-most-once, exactly once, ...



Web Services Exercise

First, perform a simple Web access using telnet

```
telnet www 80 (hit RETURN key once)
GET / HTTP/1.0 (hit RETURN key twice)
```

Second, access a well known Web service

```
http://www.xmethods.net/ve2
```

Third, write a Web Service using Java and Axis

- There are many tool choices; this one interoperates with GT3 and provides a learning path for the students
- Axis is the continuation of Apache SOAP – it includes SOAP plus Web Services API's (WSDL, UDDI,)



Writing a *Simple* Web Service

Step 1: Create the service source using Java – save as a .jws (Java Web Service) file

```
public class MyMath {  
    public int squared(int x) {  
        return x * x;  
    }  
}
```

- Unlike traditional RPC, the service code is written first, and the interface will be generated from it



Writing a *Simple* Web Service

Step 2: Use Axis tools to automatically generate the WSDL interface file

```
java -classpath $AXISCLASSPATH  
org.apache.axis.wsdl.WSDL2Java  
http://localhost:8080/axis/MyMath.jws?wsdl
```

- Can also do this step by accessing the file from a browser.
 - Axis locates the file, compiles the class, and converts SOAP calls into Java
- This example hardcodes the server pathname in the client stub

Step 3: Compile the client stubs that were generated by the previous step

```
javac -classpath $AXISCLASSPATH  
edu/uark/csce/kite/axis/MyMath_jws/*.java
```



Step 4: Write the Client Source and Compile

```
import edu.uark.csce.kite.axis.MyMath_jws.MyMathServiceLocator;
import edu.uark.csce.kite.axis.MyMath_jws.MyMathService;
import edu.uark.csce.kite.axis.MyMath_jws.MyMath;

public class MyMathClient {
    public static void main(String args[]) throws Exception {
        MyMathService service = new MyMathServiceLocator();
        MyMath myMath = service.getMyMath();
        int x = (new Integer(args[0])).intValue();
        System.out.println("The square of " + args[0] + " is " +
            myMath.squared(x));
    }
}
```

Compile with:

```
javac -classpath $AXISCLASSPATH:. MyMathClient.java
```

(Assumes that the \$AXISCLASSPATH is set up correctly.)



Web Service program execution

```
java -classpath $AXISCLASSPATH MyMathClient 4
```

The square of 4 is 16

- Axis runs in a servlet container such as Tomcat
- The Axis environment executes the server when it is called.

Pretty simple, huh? The difficulties come with the setup of the environment – more on that in a bit.



Alternatives to *Simple* Approach

Can deploy using WSDD (Web Services Deployment Descriptor) instead of using a .jws file

- Write the interface in WSDL (or write in Java and convert to WSDL)
- Use Axis or equivalent tools as before to generate stubs
- Write the implementation of the service in Java
- Write the deployment descriptor using WSDD
- Deploy on the server using the Axis admin client

Or, you can use a build file and *ant* to generate stubs and make a jar file for deployment (like the next exercise).



Grid Services Exercise

Motivation: Web services are stateless, whereas Grid services can be either stateful or stateless, and transient or persistent. Grid services also use a factory pattern to manufacture instances.

- Build on the Web Services example
- Show how to create a stateful Grid service and how Grid services differ from Web services using GT3



Grid Services Exercise Using GT3

Step 1: Define the Service *interface* using Java

```
public interface Math {  
    public void add(int a);  
    public void subtract(int a);  
    public int getValue();  
}
```

This server is stateful. The value can be modified via add or subtract, and can be accessed via getValue.

GT3 provides tools for converting the Java to WSDL



Step 2: Implement the Service

```
public class MathImpl extends GridServiceImpl implements MathPortType {  
    private int value = 0;  
  
    public MathImpl()  
    { super("Math Factory Service");  
    }  
    public void add(int a) throws RemoteException  
    { value = value + a;  
    }  
    public void subtract(int a) throws RemoteException  
    { value = value - a;  
    }  
    public int getValue() throws RemoteException  
    { return value;  
    }  
}
```



Step 3: Write the Deployment Descriptor using Web Service Deployment Descriptor (WSDD) format

```
<?xml version="1.0"?>
<deployment name="defaultServerConfig" xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="tutorial/core/factory/MathFactoryService" provider="Handler" style="wrapped">
    <parameter name="name" value="MathService Factory"/>
    <parameter name="instance-name" value="MathService Instance"/>
    <parameter name="instance-schemaPath" value="schema/gt3tutorial.core.factory/Math/MathService.wsdl"/>
    <parameter name="instance-baseClassName" value="gt3tutorial.core.factory.impl.MathImpl"/>

  <!-- Start common parameters -->
    <parameter name="allowedMethods" value="*" />
    <parameter name="persistent" value="true" />
    <parameter name="className" value="org.gridforum.ogsi.Factory"/>
    <parameter name="baseClassName" value="org.globus.ogsa.impl.ogsi.PersistentGridServiceImpl"/>
    <parameter name="schemaPath" value="schema/ogsi/ogsi_factory_service.wsdl"/>
    <parameter name="handlerClass" value="org.globus.ogsa.handlers.RPCURIPProvider"/>
    <parameter name="factoryCallback" value="org.globus.ogsa.impl.ogsi.DynamicFactoryCallbackImpl"/>
    <parameter name="operationProviders" value="org.globus.ogsa.impl.ogsi.FactoryProvider"/> </service>
</deployment>
```



Step 4: Compile and deploy the Service using *ant*

```
[aapon@kite tutorial]$ ./tutorial_build.sh gt3tutorial/core/factory/impl/Math.java
```

- You can see gar and jar files that ant creates from the source files.

```
[aapon@kite] newgrp globus
```

```
[aapon@kite] cd $GLOBUS_LOCATION
```

```
[aapon@kite] ant deploy -
```

```
  Dgar.name=/home/aapon/tutorial/build/lib/gt3tutorial.core.factory.Math.gar
```



Step 5: Write and compile the client

```
public class MathClient
{
    public static void main(String[] args)
    {
        try { // Get command-line arguments
            URL GSH = new java.net.URL(args[0]);
            int a = Integer.parseInt(args[1]);
            // Get a reference to the MathService instance
            MathServiceGridLocator myServiceLocator =
                new MathServiceGridLocator();
            MathPortType myprog = myServiceLocator.getMathService(GSH);
            // Call remote method 'add'
            myprog.add(a);
            System.out.println("Added " + a);
            // Get current value through remote method 'getValue'
            int value = myprog.getValue();
            System.out.println("Current value: " + value);
        } catch (Exception e)
        {
            System.out.println("ERROR!");
            e.printStackTrace();
        }
    }
}
```

Amy Apon, Ph.D. • University of Arkansas • May 20, 2004 • Page 21



Step 6: Start the Service and execute the client

- Start the Service

```
[aapon@kite] globus-start-container -p 8081
```

- Create the service instance. This client does not create a new instance when it runs; thus, the instance needs to be created the first time.

```
[aapon@kite] ogsi-create-service
http://localhost:8081/ogsa/services/tutorial/core/factory/MathFactoryService myprog
```

- This ogsi-create-service has two arguments: the service handle GSH and the name of the instance we want to create.
- Execute the client

```
[aapon@kite tutorial] java gt3tutorial.core.factory.client.MathClient
http://localhost:8081/ogsa/services/tutorial/core/factory/MathFactoryService/myprog 4
```

- You will see the following result: Added 4 Current value: 4

Amy Apon, Ph.D. • University of Arkansas • May 20, 2004 • Page 22



Assessment of the Exercises

- The response from the students was very favorable. Most finished the Grid exercise in about two hours. They generally felt prepared and ready to do a more complicated Grid program.
- You have to be very proficient at Java to use GT3.
- **Setup of the environment** is key to the success of the exercises



Environmental Setup

We set up GT3 first, then added the Web Services setup

Rocks cluster distribution with installed rolls

- java 3.1.0-0
- grid 3.1.0-0
- sge 3.1.0-0
- hpc 3.1.0-0



GT3 Setup for classroom exercises

- Current version of Rocks ships without the complete GT3 installation
 - Had to install the full Globus Toolkit 3
 - Future versions of Rocks will be based on NMI Toolkit version 5, which does include GT3



GT3 Setup for classroom exercises

- GT3 not set up for a multiuser environment - *ant* uses a common build directory for the deployment of all services
 - Added all users to a common “globus” group
 - Had users create unique service names
 - Had users revise the subdirectory structure of services in the user’s home directory



Web Services Setup

- Grid Services extends Web Services and uses Axis.
- However, the GT3 installation does not support Web Services development using Axis
- Had to install and configure the Tomcat servlet container to support the Axis tools for the example

Java version "1.4.2_02"

Jakarta Tomcat Ver 4.1.29

<http://jakarta.apache.org/tomcat/index.html>

Apache Axis Ver 1.1 <http://xml.apache.org/axis>



Web Services Setup

- A shell script was used to set the environmental variables so that Java can find all of the required libraries

```
#!/bin/sh
AXIS_HOME=/usr/local/axis
AXIS_LIB=$AXIS_HOME/lib
AXISCLASSPATH=$AXIS_LIB/axis.jar:$AXIS_LIB/commons-
discovery.jar:$AXIS_LIB/commons-
logging.jar:$AXIS_LIB/jaxrpc.jar:$AXIS_LIB/saaj.jar:$AXIS_LIB/log4j-
1.2.8.jar:$AXIS_LIB/xml-apis.jar: $AXIS_LIB/xercesImpl.jar:
$AXIS_LIB/wsd14j.jar:.
```

```
export AXIS_HOME AXIS_LIB AXISCLASSPATH # this is missing in the docs!!
```



Recent Changes to Grid Standards

- Introduction of Web Services Resource Framework (WSRF), January, 2004
 - Web services vendors recognized the importance of OGSi concept but would not adopt OGSi as it was defined (summer 2003)
 - Globus Alliance teamed up with Web services architects and came up with WSRF (Jan., 2004)
 - Add the ability to *create, address, inspect, discover, and manage stateful resources*
- Beta versions of GT4 are anticipated Sept., 2004



Future Work

Additional classroom exercises for Grid computing are planned!

- Improve scripts for Globus Toolkit
- Incorporation of WSRF
- Notification
- Service Discovery
- GRAM
- Access to Grid data

<http://csce.uark.edu/~aapon/grid.edu2004>
<http://csce.uark.edu/~aapon/>

