

Classroom Exercises for Grid Services

Amy Apon¹, Jens Mache², Yuriko Yara¹, and Kurt Landrus¹

¹ University of Arkansas, Fayetteville, AR

² Lewis & Clark College, Portland, OR

{aapon,yyara,kalandr}@uark.edu, jmache@lclark.edu

Abstract. Grid protocols and technologies are being adopted in a wide variety of academic, government, and industry research laboratories, and there is a growing body of research-oriented literature in Grid computing. However, there is a need for educational material that is suitable for classroom use. The goal of this paper is to develop and evaluate a suite of classroom exercises for use in a graduate or advanced undergraduate course. The exercises build on basic knowledge of operating systems concepts at the undergraduate level. This paper presents our design of the exercises. We evaluate the effectiveness of one exercise extensively and provide suggestions to educators about how to effectively use the Globus Toolkit 3 in a classroom setting.

1 Introduction

The standards for Grid computing have been developed by the Global Grid Forum [1] and include a growing body of scientific and professional publications on Grid computing [2, 3]. These standards and the toolkits are being widely adopted by research laboratories internationally. Yet, there is a need for effective classroom exercises and tools for teaching Grid computing. Our overall objectives are to identify the Grid computing topics that are appropriate to be taught to advanced undergraduates or graduate students and to present the material in a format that is suitable for classroom instruction. At the time that the project began, the toolkit in compliance with the most recent standards for Grid computing was the Globus Toolkit 3 (GT3) [4]. The specific objectives are:

- Clarify and make explicit the prerequisite knowledge to using the Globus Toolkit 3.
- Adapt materials using GT3 and additional materials from the Globus web site to laboratory exercises that are appropriate for advanced undergraduate or graduate computer science students. The adaptation will include the development of pre-laboratory exercises, learning goals for each laboratory, and post-laboratory learning assessment exercises.
- Evaluate the effectiveness of the classroom exercises that we have developed.

The Open Grid Services Architecture (OGSA) [5], released in June, 2002, is a specification of architectural issues related to broadly interoperable Grid services. The Open Grid Services Infrastructure (OGSI) [6] specifies the way in

which a client interacts with a Grid service. OGSI software provides mandatory Grid service features, such as service invocation, lifetime management, a service data interface, and security interfaces that ensure a fundamental level of interoperability among all Grid services. In January 2003, the first alpha version of the Globus Toolkit 3.0 was released [4]. The Globus Toolkit 3 is a reference implementation of OGSA and is a set of open-source software and services. GT3 builds on Web Services and the use of commodity technologies to provide an infrastructure for transforming the way on-line resources are shared across organizations.

The paper is structured as follows: Section 2 describes the classroom exercises. Section 3 is a discussion of the problems that we encountered in using the exercises in a classroom setting and our solutions to these problems. Section 4 provides the results of the student evaluations of the exercises. Finally, Section 5 describes our conclusions and future work.

2 Classroom Exercises

Our strategy in developing the classroom exercises is to utilize the Globus Toolkit 3 Tutorial [7] materials and other on-line training materials that are available at the Globus web site. The exercises described in this paper are a slight modification of what was presented to the students. Since the tools and the environment are constantly changing, the reader should consult the authors' website [8] for the most up-to-date versions of the exercises.

In addition to GT3, the teaching of Grid services requires the installation of several complementary tools. Our test platform consists of a small set of Pentium III, and Pentium IV computers interconnected with Fast Ethernet switches. The platform was installed with a fresh installation of the Rocks cluster distribution Rocks 3.1.0 [9] with the Rocks Base, the HPC, Java, SGE, and Grid Rolls installed. The Globus Toolkit 3 (version 3.0.2), Java 2 Platform Standard Edition (j2sdk-1.4.2) [10], and Apache Ant version 1.6.0 Build tools [11] were also installed. Exercise 1 also requires the installation of Jakarta Tomcat Version 4.1.29 [12] and Apache-SOAP Version 2.3.1 [13, 14]. At the time of the start of the project these versions of the tools were the latest releases that would interoperate with each other.

The following sections present details of exercises that have been developed. The exercises are designed to be done independently as an extended homework assignment by advanced students.

2.1 Exercise 1: Deployment of a Web Service

Lab Goal: Clarification and description of knowledge that is prerequisite to understanding Grid Services. At the end of this exercise students should understand the protocols used by Web services and be able to build a simple Web service.

Motivation: Grid services are built on the concept of Web services, but extend them in several ways. Although typical graduate and advanced undergraduate

students are very familiar with user-level web access and client/server systems, they may not have been exposed to the concept of Web services. Prior to this exercise we assume that they have been exposed to TCP/IP and have some understanding of programming using sockets in Java. We assume that they may not be familiar with details of Remote Procedure Call (RPC) [15,16] and some of the evolution of it, including Remote Method Invocation in both a heterogeneous language environment such as CORBA, and a homogeneous language environment such as Java. Two to three lectures on these topics precede the lab exercises, including the concepts of RPC semantics (e.g., at-most-once and exactly-once), parameter passing by value and by reference in RPC, and marshalling of parameters [17].

Preparation: Students are initially asked to read a few articles to gain background understanding that is required for the lab. Topics covered include web servers and HTTP [18, 19], Web services and SOAP [13, 20], XML [21], and WSDL [22].

1. Answer the following prelab questions about the material above:
 - (a) What is the advantage of Web Services over sockets, RPC or CORBA?
 - (b) What are the six major elements of a WSDL description?
2. Assuming that your web server is www, explain what happens if you type:

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

3. Write the Deployment Descriptor “DeploymentDescriptor.xml” to define the SOAP service. In this example our service is written using Javascript using BSF [23], and is included in the descriptor file itself. Other implementations such as java, perl or python, will need to use an appropriate provider declaration and implementation outside of the xml file.

```
<?xml version='1.0'?>
  <isd:service
    xmlns:isd='http://xml.apache.org/xml-soap/deployment'
    id='urn:xml-soap-demo-MyMath'>
    <isd:provider type='script'
      scope='Application'
      methods='gcd'>
      <isd:script language='javascript'>
        function gcd (a, b) {
          // greatest common divisor, Euclids algorithm
          int A = a;
          int B = b;
          while (B != 0) {
            int R = A % B;
            A = B;
```

```

        B = R;
    }
    return A;
}
</isd:script>
</isd:provider>
<isd:faultListener>org.apache.soap.server.DOMFaultListener
</isd:faultListener>
</isd:service>

```

4. Deploy the descriptor file using the Apache Service manager tool. Apache-SOAP provides an administration tool to manage services. There are two clients that may be used to access the service manager: an HTML client that is used via a browser, and a command-line tool. The command line tool is executed by typing:

```
java org.apache.soap.server.ServiceManagerClient.
```

To deploy a service, for example, type:

```
java org.apache.soap.server.ServiceManagerClient
http://localhost/soap/servlet/rpcrouter
deploy DeploymentDescriptor.xml
```

The service deployment can be verified using the ServiceManagerClient with the list argument as above, or through the Web interface. For example;

```
http://localhost/soap/admin/index.html
```

and click the List services button.

You will see output similar to:

```
Here are the deployed services (select one to see details)
urn:xml-soap-demo-MyMath
```

5. Write the client program GcdClient.java in Java. Example code to call the server in Java;

```

...
// Build the call.
Call call = new Call ();
call.setTargetObjectURI ('urn:xml-soap-demo-MyMath');
call.setMethodName ('gcd');
call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
Vector params = new Vector ();
params.addElement (new Parameter('arg1', String.class,
    new String (arg1), null));
params.addElement (new Parameter('arg2', String.class,
    new String (arg2), null));

```

```

call.setParams (params);

/* make the call: note that the action URI is empty because
   the XML-SOAP rpc router does not need this. This may
   change in the future. */
Url url =
    new URL ("http://localhost/soap/servlet/rpcrouter");
Response resp = call.invoke (/* router URL */ url
                             , /* actionURI */ "" );

// Check the response.
if (resp.generatedFault ()) {
    Fault fault = resp.getFault ();
    System.err.println("Generated fault: " + fault);
} else {
    Parameter result = resp.getReturnValue ();
    double res = ((Double)result.getValue ().doubleValue ());
    System.out.println("Result = " + res);
}
...

```

6. Compile and execute the client:

```

javac GcdClient.java
java GcdClient 12 35

```

7. Finally, modify GcdClient to add a method, so that prime(x) computes if x is prime.

Turn in: Your answers to the pre-lab questions and your modification of GcdClient.

2.2 Exercise 2: Deployment of a Stateful Grid Service

Lab Goal: The goal of this exercise is to understand the similarities and differences between Web Services and Grid Services. The goal is also learn to use GT3 and ant to write and deploy a stateful Grid Service. Students will learn to exploit the stateful nature of Grid Services. The exercise will consist of writing a client-server application that performs some banking transactions for a user, such as making a deposit, withdrawal, and providing a balance.

Motivation: While Grid Services have much in common with Web Services, there are also some significant differences. Web Services have instances that are stateless and non-transient. In contrast, Grid Services can be either stateful or stateless, and can be either transient or non-transient. A service instance is stateless if it cannot remember prior events, and non-transient or persistent if

the instances outlive all their clients. Unlike Web Services, Grid Services use the factory pattern to maintain multiple service instances [24]. With the factory pattern, a new instance of a selected subclass is created at run time. The factory manages the instances and allows the Grid Service to be stateful and transient. These can be divided into types of services as follows:

- *Stateful transient*: one instance is assigned to each client, thus, clients can only access their own information. Stateful means that the instance remembers what it did previously. Transient means that an instance will be destroyed sometime in the future, usually when they have served their purpose and are no longer useful.
- *Stateful non-transient*: several clients share one instance; the information in the Grid Service is available to all the clients. This approach would not be appropriate for a banking application since information should only be available to one client.

Writing a Grid Service is similar to RPC in that it starts by defining the service interface, which is also known as PortType. The interface specifies what operations will be available to the users, but it does not implement the actual operations. The language used for defining the interface is WSDL, but WSDL is not a very user-friendly language. Thus, the easiest approach to defining the interface is to write it in Java first and then generate the WSDL code using an Apache Axis tool called Java2WSDL. This approach is best suited for beginners.

Because WSDL is used to access the services, the clients of a Grid Service can be developed using any language that have bindings to WSDL. The examples and starting code use Java to develop Grid Service applications, and use a very handy build tool known as *ant*. Ant greatly simplifies the compilation and building process of Grid Services. The students are provided an appropriate build file for ant to generate the necessary executables from the source files. The build file specifies the files to be compiled, the order and the way files should be compiled. Moreover, the build file is generic so that it can be used over and over for different Grid Services, and it can be modified to suite the purpose of the programmers. All that is needed by ant is the interface description generated from a Java interface, the implementation of service written in Java, and the deployment descriptor written in WSDD.

Preparation: Grid Services are based on Web Services technologies, but they have additional features to handle the shortcomings of Web Services. Students are asked to read background material on Grid computing, including two articles by Ian Foster and Carl Kesselman [25,26], and the Globus Toolkit 3 Tutorial [7]. Students are also asked to review the basic concepts of RPC [16]. In the lab the student will build and deploy the server, build the client, and then execute the application.

1. Answer the following pre-lab questions:
 - (a) In what ways are Grid Services similar to RPC?
 - (b) In what ways do Grid Services extend the concepts of Web Services?

2. Several steps are required for setting up a user environment for the GT3 Tutorial. In summary, these steps include downloading the GT3 tutorial files and setting some environmental variables.
 - (a) In your home directory, create a directory named “tutorial” and change to this directory.
 - (b) Download, untar, and unzip the GT3 tutorial file.
 - (c) Modify the build properties file for the local installation.
 - (d) Create a new subdirectory under the tutorial subdirectory that is specific to your userid. This new subdirectory will contain the files used in the example, and will need to be modified. From the tutorial subdirectory, copy the gt3tutorial/core/factory to a subdirectory with the name “useridfactory”.
 - (e) Set the environment by running two script files:


```
source $GLOBUS_LOCATION/setenv.sh
source $GLOBUS_LOCATION/etc/globus-user-env.sh
```

3. Define the Service Interface, also known as PortType. Note that the file name can be any name, but the name must follow the correct syntax for the file and directory names. Use an Uppercase character for the first letter of your file name. For example: MyProgram.java, Math.java, Euclid.java The interface specifies what operations will be available to the users. Start with the files provided by the GT3 Tutorial and modify them to fit “userid”. In this example Java is used to create the interface. The file should be saved with the '.java' extension.

```
package useridfactory.impl;
public interface UseridBankTransaction
{
    public void deposit(float amount);
    public void withdraw(float amount);
    public amount checkbalance();
}
```

4. Write the Java code that performs the required server operations specified in the interface. The Java code will begin with the typical package, import declarations, followed by all the necessary public and/or private methods. In the actual exercise sample code is given as a start and students make modifications to it. Note that all Grid Services must extend from the base class GridServiceImpl, which is known as the skeleton class.

5. Write the Deployment Descriptor. The deployment descriptor provides the web server with information on how to publish the Grid Service. The descriptor follows the Web Service Deployment Descriptor (WSDD) format. The description includes the name (i.e., description of the Grid Service), instance-name (i.e., description of a Grid Service instance), instance-schemaPath (i.e.,

the path where the WSDL file can be found, and instance-baseClassName (i.e., base class of the Grid Service instances).

6. Deploy the Grid Service. In the deployment stage, everything is put together and made available through a Grid Service-enabled web server. All the required files, including the build file, need to be present for ant to work properly. Ant will compile the source files and generate the necessary server/client stubs and place them in a new directory, it will compile the implementation and create a Grid Archive (GAR) files. The GAR file is the actual package that will be deployed into the web server. For example:

To compile and generate all necessary files:

```
./tutorial_build.sh useridbank/impl/UseridBank.java
```

To deploy the GAR file:

```
ant deploy -Dgar.name="directory where the gar file is"}
```

7. Implement the client. The client can be a simple command-line client or a more complex and fancy GUI client. The most important part is to locate the desired service and access it with the correct syntax. Grid Services are addressed using URIs (Universal Resource Identifiers) which are called the Grid Service Handler (GSH). Each Grid Service has to have a unique GSH. The GSH tells where the Grid Service is, and the Grid Service Reference (GSR) provides information on how to communicate with the Grid Service. The client must know the GSH of the grid service. In this exercise the GSH is provided by the user as an argument, but in a future exercise Grid Location Services are used to locate the service.
8. Execute the application by starting the server and executing the client. Start the server by starting the Globus stand alone container first. From the root of the Globus installation directory run the following command:

```
globus-start-container
```

Once the server is ready, the client can connect to the server and invoke the Grid Service.

Turn In: Answers to the prelab questions, the interface definition file, and the implementation of the client and the server.

2.3 Additional Exercises Under Development

Exercise 3 will be an exercise to use the GridLocator interface and related classes to demonstrate to the students how to locate services in the Grid environment. The GridLocator interface defines the base class of all type specific ServiceLocators used to create stubs on the client side. This exercise will be comprised

of two parts, first a Web based Service Browser, which allows the students to view and explore the services defined on the Grid. It will include a sampling of predefined services which the students can examine and answer questions about. The second part will include a sample Registry service which will allow remote services to publish their GSHs and service data into a community repository of services. The students will construct a simple service, and publish it into the registry. The students will use the service browser to verify that their service has started up correctly.

Exercise 4 will be to use and understand scheduling tools for the Grid. In both cluster and grid environments, jobs are usually submitted to a scheduler which decides the order of execution of jobs depending on the scheduling algorithms being used. Students will have the opportunity to experiment with schedulers such as Portable Batch System (PBS). Globus Toolkit 3 has PBS packages that can be installed to allow interaction with local PBS schedulers.

Exercise 5 will be an exercise in MPI parallel programming using a simple master/slave paradigm in the Grid. The objective of the exercise will be to introduce students to MPI programming across the grid (composed of several heterogeneous clusters) in addition to the conventional MPI programming in a cluster. The exercise will use MPICH-G2 [27], a Grid-enabled implementation of the Message Passing Interface (MPI) standard. MPICH-G2 together with the Globus Toolkit enables the execution of scientific parallel applications across multiple machines of differing architectures.

3 Problems Encountered and Solutions

Our experiences thus far have shown us that the first essential task in teaching Grid concepts and services using classroom exercises is to have a prepared environment in which students can develop their own Grid Services. This has not turned out to be an easy task. In one of our efforts to create a development environment we installed the Rocks Cluster Distribution of Linux [9] on our small Pentium cluster and found that that distribution included only a partial version of GT3, so we had to install the full Globus Toolkit 3. This was not just a characteristic of Rocks, since we had also experienced similar missing components in other versions of Linux. We are working with the Rocks distributors to assist them in determining the correct set of packages that need to be installed to provide a complete GT3 environment.

During the testing of Exercise 2 we found that GT3 is not really set up for a multiuser compilation and deployment environment. In particular, ant uses a build file to deploy the service. During deployment, ant uses the “build” subdirectory under \$GLOBUS_LOCATION as a temporary repository. It creates new files and places them in other appropriate directories. However, the files in the build directory are owned by the user executing the command during the build process. Also a temporary directory with the default name of “gar” is created,

and removed later when ant completes successfully. Our first step was to add all student users accounts to a common “globus” group, and to change permissions on the build subdirectory so that it is group writable by everyone in the group. However, this did not work completely. If multiple users attempt to deploy an application with the same name then the deploy step will fail when files with the same name but owned by a different user are found to already exist in the final destination directory. Our additional solution to this problem was to have each student create a unique Grid Service name by prepending their userid to the interface name and the server implementation files. Then, the deploy step would write files with different names into the group writable directory.

An additional difficulty that we encountered is that some of the generated files used to undeploy a service are named using the subdirectory name in which the implementation resides. However, only the first name in the path is used. Our solution in this case was to have each student create a subdirectory with a unique name that included their userid as a substring. These name changes are reflected in the instructions for Exercise 2.

Using the current version of the ant build file that ships with the GT3 Tutorial, if ant encounters an error during deployment, it exits without removing the files and subdirectories in the build directory. Thus, other users trying to deploy their services are prevented from doing so because ant is unable to remove or overwrite the existing gar directory. In this case the gar subdirectory of the build directory is still owned by the user under which the error occurred, and write permission is not allowed on this subdirectory. A more permanent solution to these deployment problems would be to modify the build file that comes with the GT3 Tutorial for a multi-user environment.

In Exercise 2, students need to be sure that the environmental variables are set correctly to use the toolkit. If the environment variables are not set, the client may fail to execute. And finally, users need to be careful naming their files since the system is case sensitive.

4 Student Evaluations

Exercise 2 was used in the Graduate Operating Systems class at the University of Arkansas during the Spring semester, 2004. There were twelve students in the class. Feedback from the students indicate that overall our efforts were successful and the students are ready for a more challenging exercise. The average amount of time spent on Exercise 2 was two hours, although one student spent eight hours on the exercise and two students completed the exercise in only an hour.

None the students had any previous experience with Grid Services. Most students claimed a “medium” amount of experience with other, related tools, such as RPC, RMI, or Web Services. After completing the exercise, on a scale of 0 to 10, where 10 is “high”, most students felt that they had mastered the concepts in the exercise to a level of about 8. The step in which they felt the least confident was writing the deployment descriptor in WSDD.

Some specific comments from the students include:

- A second exercise would be nice.
- The exercise was really easy – only editing of code was required. I liked that it was easy and I feel like I understand each step.
- I don't understand the nuts and bolts very well and would like to go back and spend more time on each step.
- I want to write a Grid Service from scratch.
- I didn't like getting the write permission errors (from the problems when ant failed).

5 Conclusions and Future Work

We believe that courseware of this type is essential to the training of a workforce that can build the next generation Grid. The GT3 Tutorial is a very good start for someone who wants to learn about Grid Services, but modifications to the basic tutorial materials are necessary when using them in a classroom setting. This paper presents our design of classroom exercises using Web services and Grid services, the problems that we encountered in using these tools in a classroom setting, and our solutions to them.

Overall, the student response to the exercises that have been prepared so far has been excellent. We plan to complete the development of the remaining exercises and use them in courses to be taught to advanced undergraduates and graduate students during the Fall semester, 2004.

References

1. Global Grid Forum, <http://www.gridforum.org>.
2. I. Foster and C. Kesselman, *The GRID2 Blueprint for a New Computing Infrastructure*. ISBN: 1-55860-933-4: Morgan Kaufman, 2004.
3. F. Berman, G. Fox, and A. Hey, eds., *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.
4. Globus Toolkit 3, <http://www-unix.globus.org/toolkit/>.
5. I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002. <http://www.globus.org/research/papers/ogsa.pdf>.
6. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling, "Open Grid Services Infrastructure (OGSI) version 1.0," June 27 2003. <http://www.globus.org/research/papers/Final.OGSI.Specification.V1.0.pdf>.
7. B. Sotomayor. The Globus Toolkit 3 Programmers's Tutorial, <http://www.casa-sotomayor.net/gt3-tutorial/>.
8. The ARkansas Center for HIgh End Computing ARCHIE, <http://archie.csce.uark.edu>.
9. Rocks Cluster Distribution: A High Performance Linux Cluster Solution, <http://www.rocksclusters.org/Rocks/>.
10. Java 2 Platform Standard Edition, <http://java.sun.com/j2se/>.

11. Apache Ant Build Tools, <http://ant.apache.org/>.
12. Jakarta Tomcat Ver 4.1.29, <http://jakarta.apache.org/tomcat/index.html>.
13. SOAP Tutorial, <http://www.w3schools.com/soap/>.
14. Apache-SOAP v2.3.1 User's Guide, <http://ws.apache.org/soap/docs/index.html>.
15. A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.
16. A. D. Birrell and B. Nelson, "Implementing remote procedure calls," *ACM Transactions on Computer Systems*, vol. 2, no. 1, pp. 39–59, 1984.
17. P. K. Sinha, *Distributed Operating Systems: Concepts and Design*. Wiley-VCH, 1996.
18. J. Vass, J. Harwell, H. Bharadvaj, and A. Joshi, "The World Wide Web: Everything you (n)ever wanted to know about its servers," *IEEE Potentials*, pp. 33–37, Oct.,Nov. 1999.
19. Hypertext Transfer Protocol Version 1.x, <http://www.w3.org/Protocols/HTTP>.
20. Apache SOAP v2.3.1 Documentation, <http://ws.apache.org/soap/docs/>.
21. XML in 10 points, <http://www.w3.org/XML/1999/XML-in-10-points>.
22. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1, <http://www.w3.org/TR/wsdl>.
23. Bean Scripting Framework (BSF), <http://oss.software.ibm.com/developerworks/projects/bsf/>.
24. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. ISBN: 0-201-63361-2: Addison-Wesley, 1995.
25. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, 2001. <http://www.globus.org/research/papers/anatomy.pdf>.
26. I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid services for distributed system integration," *IEEE Computer*, 2002. <http://www.globus.org/research/papers/ieee-cs-2.pdf>.
27. N. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-enabled implementation of the Message Passing Interface," *Journal of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 551–563, 2004. <http://www3.niu.edu/mpi/>.