CHAPTER 3 GRID COMPUTING INFRASTRUCTURE SOFTWARE

CONTENTS

3.1 Standardization Bodies

3.2 Interacting Grid Computing Components
   Development of a Service Oriented Approach
   Stateful Web Services
   Transient Services

3.3 Open Grid Services Architecture (OGSA)
   Purpose
   Open Grid Services Infrastructure (OGSI)
   WS-Resource Framework
   Generic Stateful WSRF Service
   More Advanced Features of WSRF/GT4 services incomplete

3.4 After WSRF incomplete
   Representational State Transfer RESTful Web Services

3.5 Summary

Further Reading
Bibliography
Self-Assessment Questions
Programming Assignments

Oct 10, 2008

Do not distribute.

Copyright B. Wilkinson

This material is the property of Professor Barry Wilkinson and is for the sole and exclusive use of the students enrolled in the Fall 2008 Grid computing course broadcast on the North Carolina Research and Education Network (NCREN) to universities across North Carolina.
In this chapter, we begin our presentation on Grid computing infrastructure software. In Chapter 2, we outlined Web service technology. Now we will move onto how that technology is applied to Grid computing, which includes the key question of how state should be implemented. A critical aspect is that standardized approaches are adopted, and we will describe effects towards standardization. This is an ongoing process with continual improvements and refinements. We will continue our discussion of the Globus toolkit and in particular its use of Web service technology.

3.1 STANDARDIZATION BODIES

Standardization is critical for the wide-scale adoption of Grid computing. The underlying network for Grid Computing has become the Internet rather than customized dedicated network for the most part. Even if a dedicated network is used, the network uses Internet Protocols. A number of standardization bodies have come into existence for standardizing Internet and World-Wide Web protocols. IETF (Internet Engineering Task Force) was formed in 1985 for Internet standards, including the previously developed TCP/IP protocol. Other standardization bodies appeared in the early 1990’s with the ever growing Internet. W3C (World Wide Web consortium) was founded by Tim Berners-Lee shortly after he conceived the World Wide Web in the early 1990’s. W3C works on standardization of Web-related technologies including XML. Other Internet related standardization bodies include OASIS (Organization for
the Advancement of Structured Information Standards). OASIS began as SGML Open in 1993 to promote the SGML mark-up language and became OASIS in 1998 to focus on structured information standards including newly developed XML mark-up language. DMTF (Distributed Management Task Force) was created in 1992 for IT systems management infrastructure.

The Grid community started to form its own special interest bodies for adopting Grid computing standards in the late 1990s. In the US, the Grid Forum was formed in 1998. Later, this was merged with eGrid (European Grid) forum and the Asia-Pacific Grid community to form the international Global Grid Forum (GGF), which has become the central forum for discussing and developing Grid computing standards. GGF has three meeting each year, numbering from the first meeting (GGF1 in Amsterdam in March 2001). GGF became the Open Grid Forum (OGF) in 2006, with a merger with an industry group called Enterprise Grid Alliance (EGA) and continued the tri-yearly meetings, GGF 18 becoming OGF 18. OGF 24 took place in September 2008 in Singapore. These meetings have grown to a very large endeavour today, with participants across the world attending.

Once Web services were introduced, many standards began to be developed for use in Web service environments, most notably by W3C organization. Web services exploit XML, which had been introduced in 1998, and SOAP, which had been ratified in 2000. Subsequent standards continued to be developed including WSDL described in Chapter 2, and large family of WS-* standards where * refers to the name of one of many standards. These standards were not developed specifically for Grid computing but it became natural to use them for a Grid computing environment with the adoption of Web services in Grid computing. Web services provide an easily identifiable interface through WDSL documents and conveniently use Internet addressing (URLs). However, there are some issues with using Web services for Grid computing as we shall now address.

### 3.2 INTERACTING GRID COMPUTING COMPONENTS

#### 3.2.1 Development of a Service Oriented Approach

Grid computing infrastructure needs to address all aspects of operating a Grid computing environment including how security, job submission, file transfers, and information services are implemented. Individual components need to be provided for each area with multiple interacting complex components. How Grid computing infrastructure should be implemented has been evolving since its introduction in the mid-1990’s. It has gone through several development cycles, which have still not finished and may never do so as dissatisfaction is found in each approach. Early Grid computing work focused on customized solutions, that is, projects developed in-house software to achieve Grid computing facilities. Some projects concentrated upon using a specific approach such as using an object-oriented approach. As described in Chapter 1, the Globus project started after the early I-Way experiment on Grid computing in 1995, and developed a software toolkit that has progressed through several versions. The Globus toolkit version 2 (GT 2) introduced in 2002
became widely adopted by the Grid community and provided a suite of software tools (APIs) for building a Grid environment. GT 2 became a de facto standard and very highly regarded but it was not based upon any industry wide standards. Many early Grid computing projects in the early 2000’s used GT 2.x. For example, our undergraduate Grid computing course offered in 2004 and other Grid computing courses of that era used GT 2.x.

The early work of developing Grid computing infrastructure tools did not use Web services described in Chapter 2 as that work predated Web services as we know it, which came into being later. The early experiences provided the impetus for later standardized work. The use of standardized technologies is critical to the wide adoption of Grid computing. It was now been recognized that Web services provide the way forward for Grid computing to provide a uniform interface for Grid Computing components especially those that need access through the Internet. A key aspect of Grid services is XML, which provides a unified platform-independent way of describing information for Web services and generally.

### 3.2.2 Stateful Web Services

Web services are usually stateless. They do not remember information from one invocation to the next. That works fine in many Web service applications. They do not need to know what happened with previous invocation by another client, nor should they. A simple example of a stateless Web service is the Web service introduced in Chapter 2 that performs a function using as input the arguments to the Web service and returning a value only based upon input arguments and the function. For example a Web service, MathService, might a method to square an argument and return its value as illustrated in Figure 3.1.

A stateful Web service has the ability to remember information from one invocation to the next and between invocations by different clients. This would be most useful in Grid computing applications to enable a sequence of actions to be done dependent upon past actions and results. A stateful Web service is illustrated in Figure 3.2. In this example, the Web service performs the add operation on a variable called data inside the Web service, and its value is retained between invocations by clients, the same client or different clients. Each client has access to the value of data. The figure shows a client making a request to add 3 to data, which will
become 7 if its original value was 4. Suppose there is another method called `getValue` that returns the current value of `data`. If the same or another client accesses the Web service using that method, it will get the value 7.

Clearly the previous example is artificially simple although it is useful to explain the concept. It occurs in the book by Sotomayor and Childers (2006) in the context of implementing Globus 4.0 services and can be the basis of Grid computing assignments for creating Globus-hosted services. A real example of a stateful Web service would be one in which access a large database as illustrated in Figure 3.3. Here a client make a request to access data from a large persistent database through a stateless Web service. The result of the request, that is the required stored data, is returned to the client through the Web service. The Web service could provide other methods such as transferring data from one database to another database. A key aspect here is that the state is contained in a resource that is separate from the Web service. The Web service becomes a front-end to the resource (or resources). We shall come back to this concept later.

### 3.2.3 Transient Services

Another feature originally thought to be required for Grid Computing is the ability to make service transient, that is, a service that can be created and destroyed. Web services are non-transient and generally always available for clients. They can be accessed by different clients on a need basis. They do not have the concept of
dynamic service creation and destruction. Transient services, on the other hand, are created and potentially can be destroyed on a need basis. Typically, although not necessarily, transient services would be created by specific clients and do not outlive their clients. Rather than make the service transient, one can make the stateful resource associated with the service transient or created dynamically. We will look at how transient services/resources might be created. In general, it has led to a somewhat object-oriented approach where instances are created using a factory service. In Globus 3, instances of the actual services were created while in Globus 4, only instances of the associated resources were created, not the services themselves. In any event, dynamic creation means that matters such as their lifetime also need to be addressed.

3.3 OPEN GRID SERVICES ARCHITECTURE (OGSA)

3.3.1 Purpose

Open Grid Services Architecture (OGSA) was originally proposed by Foster et al. in the seminal paper “The Physiology of the Grid” (2002) and was introduced at the Global Grid Forum GGF4 meeting in February 2002 as a Grid computing standard. OGSA defines standard mechanisms for creating, naming, and discovering service instances and addresses architectural issues relating to interoperable services for Grid computing. OGSA attempts to address all the fundamental services of Grid computing such as for job management, resource management, security services, and service discovery. It specifies standard interfaces for these services, that is, what each service should provide but it does not give implementation details. OGSA requires stateful services but does not say how that will be achieved. Following the traditional approach to system design, such implementation details are left to other standards.

3.3.2 Open Grid Services Infrastructure (OGSI)

The first attempt by the Grid computing community to specify how OGSA could be implemented was the Open Grid Services Infrastructure (OGSI) standard, which was introduced in 2002-3 (final draft in 2003). OGSI specifies the way that clients interact with services (that is, service invocation, management of data, security mechanism, etc.). The approach taken in OGSI to implement a stateful Web service was to modify the Web Service Description Language, WSDL, to enable state to be specified. The modified language was called Grid Web Service Definition Language (GWSDL).\(^1\) OGSI also introduced and described the term Grid service as an extended Web service that conforms to its OGSI standard. GWSDL provided support for the extra features in Grid services that were not present in Web services. In addition to a means of representing state, OGSI included inheritance of portTypes (interfaces), a way of addressing services called Grid Service References, notifications, and a point-to-

\(^1\) The word “definition” from WSDL then being for Web Service Definition Language rather than Web Service Description Language.
point message passing mechanism. More details of OGSI can be found at Tuecke et al. (2003).

With the appearance of OGSI, the Globus team moved onto implementing the OSGI standard, and released Globus version 3 in 2003. Version 3 was a complete implementation of OGSI and provided additional Web services, some built onto top of OGSI. A few Grids computing projects migrated from Globus version 2 to Globus version 3, including our undergraduate Grid computing course in 2004-5, but it soon became apparent that there were significant issues with OGSI. Within a year or so, the Grid community had found the approach not acceptable because:

- It significantly modified pure Web services
- It required new tools. GWSDL required tools in addition to that used for Web services.
- It was too object oriented in approach.
- There was too much specified in one standard. It would be better broken down in a series of specifications.

A better way was needed.

### 3.3.3 WS-Resource Framework

While the Grid community had been exploring ways to embody state in Web services for Grid computing environments, the Web service community had also been working on approaches to stateful Web services. The two communities began to merge on an approach for stateful Web services. A group that included several industry partners (Computer Associates, Fujitsu, Hewlett-Packard, IBM) and the Globus Alliance (a group associated with Globus established in Sept 2003 for developing Grid technologies) proposed a specification called WS-Resource Framework (WSRF) in 2004. It was ratified by OASIS essentially to replace OGSI and make the implementation of a stateful Web service acceptable to both the Web services and Grid services communities. WSRF specifies how to make a Web service stateful and other features, without drifting too far from the original Web service concept.

The term Grid service, which originally was introduced with OGSI, continued for while. A broad meaning of Grid service is any service that conforms to interface conventions of a Grid computing infrastructure. A narrow WSRF meaning is service that conforms to WSRF. The term seems to have lost favor, maybe because it better to think of services in a Grid environment simply as a regular Web service. OGSA also continues. OGSA requires a stateful Web service. WSRF specifies how that stateful Web service is implemented. Stateful Web services are extensions of the original stateless Web service.

A stateful Web service is obtained in WSRF by having a Web service front-end to stateful “resources”. Previously, the stateful resources were closely linked to the Web service in the GWSDL specification. Now the two are clearly separated, as illustrated in Figure 3.4. In this example, the resources consist of two WSRF “resource properties”, which are persistent values that are retained between invocations. The service interface is described in WSDL. The WSDL file serves the same purpose as
in the original stateless Web service, i.e. to define the service interface. Using WSDL allows existing WSDL parsing tools to be used. A significant addition in the WSDL file is a specification of the resources. In WSRF, the Web service is described in a WSDL document and the resource is specified in a separate Resource Properties document (or merged into the WDSL document in the case of simple resources). A *WS-Resource* is a Web service and an associated resource.

Let us revisit the stateful Web service example shown previously in Figure 3.3. Figure 3.5 shows a WSRF version of this stateful Web service. The `add` method adds an integer value to a variable called `data`. If a service implements operations on a WSRF resource properties, WSDL will include definitions relating to the resource property.

---

**Figure 3.4** Web Service Resource Framework (WSRF).

**Figure 3.5** WSRF Stateful Web service.
**WSRF Addressing - End Point References.** Each WSRF service needs an addressing mechanism that includes addressing to the resources. Pure stateless Web services are addressed with URIs (Uniform Resource Identifiers). Typically, they are addressed by URLs (Uniform Resource Locators) as generally used for Web sites. URLs are a subset of URIs. The previous OGSI standard used an address called a *Grid Service Handle*, which uniquely identified each service. The WSRF service addressing mechanism is defined in the *WS-addressing* standard and uses a term called an *End-Point Reference* (EPR), which is an XML document that contains various information about the service and resource. Specifically, the End Point Reference includes both Service address (URI) and a resource identification called a *key*. The key is a number. The service that accesses the resources and the resources are paired together. The client makes an request to the (stateless) Web service, addressing the service using the EPR. The URI part of the EPR identifies the service. The web service selects the resource using the resource key inside the EPR, as illustrated in Figure 3.6.

An End Point Reference has required and optional entries. An End Point Reference could be used simply to address a Web service without an associated resource. In that case, the EPR would have the form:

```xml
<wsa:EndpointReference>
  <wsa:Address>
    http://www.cs.uncc.edu/myWebService
  </wsa:Address>
</wsa:EndpointReference>
```

where the `<Address>` tag is required, the prefix `wsa` is the WS-Addressing namespace (`http://www.w3.org/2005/08/addressing/wsdl`), and `http://www.cs.uncc.edu/myWebService` is the URL of the Web service.

With the inclusion of a resource (i.e. WS-Resource), an EPR could be used to address the Web service and an associated resource. The following is an example:

```xml
<wsa:EndpointReference>
  <wsa:Address>
    http://www.cs.uncc.edu/myWebService
  </wsa:Address>
  
  <wsa:ReferenceParameters>
    <wsa:KeyInParameter>123</wsa:KeyInParameter>
  </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

![Figure 3.6 End Point Reference (EPR).](image)
The resource identification is specified within the `<ReferenceParameters>` element. Note that the resource identifier is not specified within the WSDL document itself. It is separated out and obtained by a separate mechanism when the EPR is obtained through an index service or service provider. The resource identifier will appear in the header part of the SOAP message rather than in the body. More information on this can be found in Foster et al. (2005).

**WSRF Specifications.** WSRF is actually a collection of four specifications (standards):

- **WS-ResourceProperties** - specifies how resource properties are defined and accessed
- **WS-ResourceLifetime** - specifies mechanisms to manage resource lifetimes
- **WS-ServiceGroup** - specifies how to group services or WS-Resources together
- **WS-BaseFaults** - specifies how to report faults

Additional WS-* standards include:

- **WS-Notification** - collection of specifications that specifies how to configure services as notification producers or consumers
- **WS-Addressing** - specifies how to address Web services. Provides a way to address a Web service/resource pair. Defines an endpoint reference.

### 3.3.4 Generic Stateful WSRF Service

We will briefly describe how to program a simple WSRF service. The following section is based upon detailed code provided by Sotomayor and Childers (2006). Here, we have simplified the code, including not having specific mathematical or get operations or a `<types>` element for defining the data format of the messages. However, as we shall see, a `<types>` element will be still needed to define the resource properties. The reader is referred to Sotomayor and Childers (2006) for a more complete discussion of stateful Globus 4 services.

**WSDL code.** For the purposes of this section, we will describe the functionality in general terms, as we did in Chapter 2. In Chapter 2, we outlines a generic Web service perform a function that has one integer argument supplied, `arg1`, and returns an integer `result` based only upon the supplied argument. Now, we will extend that simple Web service so that the function performed uses a stored integer, `data`, as illustrated in Figure 3.7. The actual function is still undefined at the WSDL level.
The format of the stateless service WSDL 1.1 document was given in Chapter 2, Section 2.2.3, and this carries over for a WSRF WSDL document. But in addition, the properties of the resources need to be specified. The WSDL code for a stateful service is shown in Figure 3.8. Compared to the stateless service code given in Figure 2.10, the additions are resource properties and a namespace for use with the resource properties.

**Resource Properties.** Resource properties are specified in the `<types>` element:

```
<types>
  <!-- RESOURCE PROPERTIES -->
  <xsd:element name="data" type="xsd:int"/>
  <xsd:element name="FunctionResourceProperties">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:data1" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</types>
```

declares data, as occurring just once.

**PortType.** The portype is the same as the stateless version except that ResourceProperties is included as an attribute:

```
<!-- PORTTYPE -->
<portType name="FunctionPortType"
```

![Figure 3.7 Generic stateful Web service with a WSDL document and a stored value.](image-url)
Figure 3.8  WSDL code. Based upon Sotomayor and Childers 2006. CHECK/Alter
Bindings. In a traditional WSDL document (Chapter 2), bindings provide how abstract interface maps to concrete protocol messages. This is not needed in Globus 4.x systems because bindings are automatically inserted by Globus when the service is built using Globus tools. Hence, there is no <binding> element in the code. Also there is no service element providing a network address in the presented code.

Service Code. The WSRF service has two major parts:

- Resource properties
- Service code (methods)

For simple service and resources such as the first WSRF service described in Sotomajor and Childers (2006, Chapter 6), the service methods and resources properties can be combined into one file, in essence service code that consists of:

```java
public class MyFunct {
    private int data // resource property

    public int funct1(int x){
        int f = .. data .. ; // compute some function of x and data
        return f;
    }
}
```

The actual code is somewhat more complex as it has to incorporate a GT 4 class that handle the way that the resource property is handled, which is outside the scope of this book. Sotomayor and Childers 2006 provide a full treatment of the different ways resources properties can be handled.

Deploying GT 4 Services. Globus version 4 uses Apache Axis Web service container internally and the basic steps for deploying a GT4 service is similar in concept to that described for Apache Axis in Chapter 2. The package of files needed for deployment are the WSDL service interface file, the service code, WSDD (Web Service Deployment Descriptor) deployment file, the JNDI (Java Naming Directory Interface) configuration file, and GT4 build files. The package can be build using the ant (Another Neat Tool) build tool, creating a so-called gar file (Grid archive). Globus contains a build script globus-build-service.sh that contains the ant and bash commands to build the service. A Windows version in Python is provided.
Once the gar file has been created, the Globus tool `globus-deploy-gar` is used to deploy the service. Then, the GT4 container has to be started/re-started to complete the process and make the service accessible. The GT4 container is started with the command `globus-start-container`. The option `-nosec` will start the container with no security, that is, using the http protocol rather than the https protocol, and without the needing for certificates. Figure 3.9 shows a sample output from the command `globus-start-container -nosec`. In this example, the host computer has the Internet address 166.82.130.77. Running a container only requires the Globus “core” common runtime component. Versions of the Globus core are provided in Java, C and Python, Java being the most common. The Java core can be installed and services deployed used on any machine with JDK.

Deploying your own Globus service is useful to understand the inner workings of Globus components, and potentially to create your own Globus service based application. It is possible for users to start a separate container for themselves on a server. However, each container has a rather large footprint and it is impractical to have a large number of separate containers running simultaneously (Operating system thrashing occurs). Also when a container is started, it will deploy all available services. Hence one would see the deployable services of everyone within your container. Each service must have a unique name. In the our Fall 2005 Grid computing course, a script was written to automatically rename students’ services but still problems can occur that cause all work in the whole class to be held up. Hence a different approach was used, that of installing GT4 core on personal computers. Without an external Internet connection, one will get the local host address (e.g. 192.168.123.100…) or a local network address (e.g. 127.0.0.1) displayed in Figure 3.9.

![Figure 3.9 Globus 4.0 container with services.](image-url)
Once a service is deployed in a running GT4 container, it can be used by a client in much the same way as a client for stateless Web service in Chapter 2. A URI will be used to locate the services, for example \texttt{http://166.82.130.77:8080/wsrfservices/WidgetService} in Figure 3.9. With the resource as part of the service, the URI is sufficient. In the case of a resource that has been physically separated from the service, but the URI is part of an EPR with key locate the resource.

### 3.3.5 More Advanced Features of WSRF/GT4 services

In this section we will very briefly outline some of the many features of WSRF services ...

### 3.4 AFTER WSRF

Grid Computing software development is a moving target. WSRF was developed to overcome the previous OGSI/GWSDL attempt at defining a stateful Web service. But even WSRF has its critics and we can expect more developments ..... 

### 3.4.1 Representational State Transfer RESTful Web Services

To add

### 3.5 SUMMARY

This chapter introduced the following concepts:

- Stateful Web services
- Transient Web services
- Globus components
- WSRF service implementation

### FURTHER READING

The principal source for coding Globus Toolkit WSRF services is Sotomajor and Childers (2006). This book provides clear and very detailed account on how to code such services with many complete code examples. Details of coding WSRF services not in the context of Globus can be found in Graham et al. (2005)
BIBLIOGRAPHY


SELF-ASSESSMENT QUESTIONS

The following questions are multiple choice. Unless otherwise noted, there is only one correct question for each question.

1. What is GWSDL?
   a) Global Web Service Description Language
   b) Grid Web Service Description Language
   c) Google’s Web Service Description Language
   d) Grid Wide Service Description Language

2. What is meant by the term portType? CH 2
   a) A specific port chosen to be used by containers
   b) An abstract definition of the operation of a service
   c) The types of ports available to be used
d) The type of data passed through a port

3. What is a non-transient service?
   a) An instance of a service that does not receive data
   b) An instance of a service that outlives its client
   c) An instance of a service that generates stateful data
   d) An instance of a service that generates stable data

4. What is a factory service?
   a) A service designed to build products
   b) A service designed to create Grid services
   c) A service designed to create instances of Grid services
   d) None of the other answers

5. Name one basic difference between a Web service and a Grid service
   a) None
   b) A Grid service is accessed from a Grid
   c) A Grid service can be stateful
   d) The connections between Web services are in the form of a Web and the
      connections between Grid services are in the form of a Grid.

6. How do you close your container in GT 4? (Assignment 2) unfinished/???
   a) globus-start-container soft ....
   b) globus-stop-container soft ....
   c) Close your window
   d) rm container

7. What is a (OGSI) notification in Grid services?
   a) A final demand to pay your taxes.
   b) A mechanism for Grid service to inform a client of changes in the service
   c) A mechanism for client to inform a Grid service of its existence
   d) A mechanism of the GGF committee to inform the community of changes to the
      (OGSI) standard

8. Name one program (tool) is used to deploy a GT 4 service. (Assignment 3 check)
   a) ant
   b) condor_submit
   c) build.sh
   d) java

9. How does one determine what operations can be invoked on a service?
   a) Look at the WSDL interface document
   b) Guess
   c) Each operation is given a predefined name agreed to by a standard of W3C
   d) Look at additional documentation regarding service

10. How are Web services addressed in Globus 4.0?
a) By programmer-defined grid service names.
b) By URI’s
c) By an End Point Reference.
d) By port numbers.

PROBLEMS

???