Writing Grid Service Using GT3 Core

Dec, 2003

Long Wang
wangling@mail.utexas.edu
Department of Electrical & Computer Engineering
The University of Texas at Austin

James C. Browne
browne@cs.utexas.edu
Department of Computer Science
The University of Texas at Austin

Abstract

OGSA (Open Grid Services Architecture) is proposed in 2002 which integrates the Globus standard with the commercial Web Service standard, provides a unified and standard architecture for grid computing. Grid service is the central concept of OGSA, which essentially is web service with improved functionalities. GT3 (Globus Toolkit 3.0) is the first implementation of OGSI (Open Grid Service Infrastructure) Specification 1.0. We show the usefulness of Grid Service by introducing a demo application based on GT3 core, and the step-by-step procedure of building Grid Service based application.

Key Words: GT3, Grid Service

1 Introduction

Grid computing technology is being developed to solve two kinds of problems. First, there is much resource wasting in the internet. Such resources include processing cycles, disk space, data and network. Second, the integration of different systems deployed in a large company tends to be difficult. We need standard technology and platform to support such integration.

To solve this problem, grid computing considers all the available resource in the network as a "super computer". User can transparently use and manage all these resources. Grid computing also provides a series of standard to integrate heterogeneous systems.

In 2002, OGSA (Open Grid Service Architecture) is proposed which integrates the Globus stand with the Web Service standard, and is going to be the unified standard for the grid computing. The basic concept of OGSA is grid service which is essentially a web service with improved functionalities and behaviors.

Web service is selected because compared to other distributed computing technologies, such as CORBA, RMI and EJB, it is a more suitable candidate for internet scale application. First, web service is based on a collection of open standards, such as XML, SOAP, WSDL and UDDI. It is platform independent and programming language independent because it uses standard XML language. Second, web service uses HTTP as the communication protocol. That is a big advantage because most of the Internet's proxies and firewalls will not mess with HTTP traffic.

However, web service is not powerful to build complex applications. It lacks some functionality, such as lifecycle management, notification and persistency. And web service is stateless which means it can't remember what has been done from one invocation to another.

Grid service provides more versatile functionality than web service. We will cover these functionalities in section 2. In section 3, we will use a demo application to introduce how to develop grid service application by using GT3 core. In section 4, we will evaluate the grid service and GT3 core in terms of performance, functionality and ease of use. Section 5 will concludes the report and describes the future work.

2 Grid Service

2.1 Persistent Grid Service and Transient Service Instance

Instead of having a single stateless web service instance shared by all the users. Grid service makes use of factory approach. For any grid service, a factory will be generated when this service is deployed in the server the first time. And this factory will in charge of all the service instances.

For example, when a user needs to be served by a new service instance, it will talk to the factory and ask it to generate a new instance. Whenever a new instance is created, a global unique handle (GSH, Grid Service Handle) will be assigned to it. GSH is just like an address which will specify where the grid service is. To find out how to communicate with the grid service, the GSH is sent to the handle resolver, and the GSR (Grid Service Reference) will be returned which tells detail information about grid service, such as methods and message types. If SOAP binding is used, such GSR is just a WSDL file.

Each service instance is transient, and its lifecycle can be managed by the users. We will talk about the lifecycle management later. In this way, different users can talk to individual service instance and will not mess with each other. Actually, one service instance can also serve multiple different users when needed.

2.2 Service Data

We know in uddi directory, we can discover the service that can satisfy our requirements. But if there are many similar services, how can we know which service is most suitable. One possible solution is use WSDL since it is the language used to describe service. However, there are two problems using WSDL. First, it is too technical and not easy to query. Second, WSDL file is not easy to be modified dynamically if these service data need frequent change.

Service Data is a structured collection of information that is associated to a grid service. Each service instance has its own service data set which includes one or multiple SDE (service data elements). A SDE can be a simple name-value pare, or a complex data structure.

In the real implementation in GT3 core, each SDE is defined as a Java class, just like a Java Bean. We can use the similar methods to set and get the values of SDE. A SDD (service data description) file is used to define the data structure of the SDE which will be sent to client with the WSDL file so that client will know what kind of the service data this service has.

2.3 Lifecycle Management

Since each service instance is transient, there needs to be some lifecycle management mechanisms. The lifecycle of the service instance refers to the time between instance creation and destruction. Some service instance may live as long as server, that means they will be loaded whenever the server restarts.

There are two approaches provided by GT3 core to define the lifecycle of a service instance. One is to negotiate with service factory about the lifecycle before creating the instance. The other is to send some message to the service instance in the middle of its lifecycle. Such messages include requestTerminationBefore and requestTerminationAfter.

GT3 core provides some callback methods to allow clients get notified when lifecycle is changed. Lifecycle monitor is also provided to help users manage lifecycles.

2.4 Notification

Notifications allow clients to be notified of changes that occur in a grid service. Here the changes refer to service data. The inclusion of notification mechanism allows different parts of the program or different users to communicate with each other through service data. Web service does not provide notification because it is stateless.

User can subscribe and begin listening. Whenever the service data is changed, these subscribed users will be notified immediately. There are two types of notification: pull and push. For the pull type, service instance will just tell the user that change happens, and the user need to make another call to the service instance to get the updated service data after notification. For the push type, the updated information comes with the notification.

In GT3 core, user does not subscribe to the service instance, but to each service data element. This fine-grained notification may reduce network traffic and improve the system performance.

3 Developing Grid Service Application

3.1 Programming Model

The programming model of GT3 core is just the same as ordinary distributed computing model: proxy-stub. The application includes two parts, the server side and the client side. These two sides are loosely coupled. They connect with each other through WSDL file. Developer must provide a WSDL file to describe the service interface, message types and binding mechanisms. Client can generate stubs after receiving WSDL file and call the service instance through stubs.

Developers can directly write WSDL files. (Actually, this is called GWSDL file in GT3 core. WSDL 1.0 does not support inheritance and some other functionality. They will be included in the WSDL 2.0 specification. For now, the GWSDL file will be transferred to regular WSDL file to support legacy systems.) This approach may be a little complex for those who are not familiar with web service. However, it is powerful to support large scale grid service applications. The other approach is to generate WSDL files from the Java interface or IDL interface. GT3 core provides some tools, such as Java2WSDL to help such a transition task. This approach is simple since developers don't need to understand the grammar of WSDL. Its shortcoming is some interfaces can not be correctly transferred to WSDL files with existing files.

Writing a grid service application is quite hard and is not straightforward for those beginners. So GT3 core provides many tools to help us developing and deploying grid service application. Ant is also used to reduce the burden of remembering the names of bunch of tools. We will use a demo application to describe the developing and deploying procedures in the following part of this section.

3.2 Demo Application Development

This is a simple online auction application which simulates an English auction market. The auctioneer will create an auction service, and different users can submit their bids to this market. If the submitted price is higher than the current highest price, the highest price and the winner will be changed. Users can also subscribe to the service data of this auction service. Whenever the highest price and other bidding information are changed, they will get notified.

Here we specify the interface by writing the GWSDL directly. It is almost the same as writing a WSDL file except two main differences. The first is, GWSDL allows inheritance. The portType here inherits from two portTypes provided by GT3 Core, GridService and NotificationSource. So we only need to add some application specific operations and service data. The second difference is the inclusion of service data. We define a new data structure named AuctionDataType in an XSD file. All the state information about this auction service, such as highest price and current winner, will be stored as service data.

After having the interface files, we need to implement the grid service. There is not much difference between writing a grid service implementation and other java programs. Your implementation object needs to directly extend the GridService or implement the

OperationProvider interface. (There will be some difference between these two methods when you writing the deployment descriptor.) If there is service data come with the grid service, you need to initialize the service data. If you want to add some lifecycle management functionality, you can just implement the GridServiceCallback interface. In a word, the implementation work is very simple.

The next thing to do is to write deployment descriptor which is actually a WSDD file. You need to specify some parameters about the service you are going to deployed. These parameters may include the service name, the location of interface and implementation instance. You can also specify which methods are allowed to be invoked and whether the service will be reloaded after the web server restarts.

To compile the source code, you need to use Ant. Otherwise you will have to remember all the tools names and the compiling sequence. To generate the java interface from the GWSDL file, you need to specify the mappings from the namespace to the package names. This is done by writing a namespace2package.mappings file. The compiling results will include the stubs, the WSDL file and all the other binary code. All this stuff will be add to an archive which is a GAR file. GT3 core will deploy the grid service by extracting this file archive and putting files under the correct directory and modify the system configuration files.

After correctly deployment, you will see find the corresponding factory service after the grid container starts. For this application, factory service is named AuctionFactoryService, and its location is http://localhost:8080/ogsa/services/demo/cs395t/AuctionFactoryService, you can create a auction service by invoking such factory service.

The final work is to write some client code to test the auction service. The client is able to submit bid by calling the remote method. And after subscription to the service data, he will be notified whenever the service state is changed.

4 Evaluation of Grid Service and GT3 Core

The combination of grid computing with the commercial web service provides a standard infrastructure for existing heterogeneous distributing systems. As the central concept of this infrastructure, the grid service already shows great power to support large scale internet-based applications. Based on web service, grid service is platform independent and language independent and supports service discovery by looking up in the uddi directory. And it adds

more versatile functionalities such as transient service instance, soft lifecycle management, service data and notification. These powerful features make it suitable candidate for more complex and larger grid applications which web service can not support.

As the first implementation of OGSI, GT3 core already shows the power of the grid service. And it provides many tools to make the application development much easier. The developing and deploying procedures are straightforward and easy to understand. Some tools, such Java2WSDL, provide some support to the legacy java applications and easier for those developers who are unfamiliar with the web service.

However, since both OGSA and GT3 are very new, there are many problems remains to be solved and some bug to be fixed. Some are listed as follows,

- The overhead of remote procedure call is still high. For my demo application, although the client and the server locate in the same machine, the delay is very long. So, one practical problem is how to ensure the quality of service. This will be especially important for those time critical enterprise applications.
- WSDL is not concise and prone to error. Nobody likes to read the WSDL file to understand the service interface. We need some tools to provide more user-friendly interface design functionality. And these tools must support grammar checking and type checking to reduce the burden of developers.
- Since the service name must be global unique, it is inconvenient to application debugging and upgrading. Imagine you have an early version of the application named auction, and you find some bugs and want to deploy another version, however, you will find out that you can not name it as auction. You have to explicitly undeploy the first version to get the second version work.
- The authorization mechanism is not fine-grained.
- It is not able to modify the grid service dynamically when the service has been deployed and the grid container is on.

It turns out there is still a long way to go to improve the system performance.

5 Conclusion

As the key concept of OGSA, grid service is powerful and flexible enough to support grid computing and is able to provide the standard technology and platform for heterogeneous systems. GT3 core fully implements OGSI, provides many tools to make programming with grid service much easy. We develop an online auction grid service to show the power of GT3 core. Although there remains some inconvenient design and bugs, it is an excellent toolkit to write grid service.

The grid service and GT3 are new and lots of work to be done. The most important is to reduce the remote call overhead. Type checking, ease of use and dynamically deployment are possible goals need to achieve in the future.

6 References

[1] Foster, C. Kesselman, J. Nick, S. Tuecke: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Global Grid Forum, 2002

[2] S. Tuecke, K. Czajkowski, I. Foster, etc: Open Grid Services Infrastructure (OGSI) Version 1.0, Global Grid Forum Draft Recommendation, 2003

[3] http://www.globus.org

[4] http://www.casa-sotomayor.net/gt3-tutorial