OGSA-DAI WSRF Tutorial: The Client Toolkit

NGS Induction Event,
Guy Warner, NeSC Training Team

This OGSA-DAI WSRF tutorial shows how to use the OGSA-DAI client toolkit to access a data source and perform multiple actions on the data before retrieving the results.

Initially a database that is running on the host "tc03.nesc.ed.ac.uk" is accessed. This database contains an example address book with fictional names and addresses. A copy of the WSRF version of OGSA-DAI (2.1) has been installed and configured to access this database. OGSA-DAI in turn is using the Globus container (the Globus equivalent of Apache Tomcat) as the means of exposing the service on to the internet. The database is running under PostgreSQL.

A second database will also be used towards the end of this tutorial. This database is running on the host "tc05.nesc.ed.ac.uk". The database this time has a different schema to the database on "tc03". This database contains a table called stafflist, an example list of staff with (again somewhat) fictional entries. In fact there are several other databases hosted on these servers. These databases are used for providing space for temporary tables and for the exercise at the end of this tutorial.

These initially involve accessing the LittleBlackBook database via tc03.nesc.ed.ac.uk that was used in the previous tutorial.

Because the tool used in this part of the tutorial can end up consuming large amounts of memory it is neccessary to spread the load across multiple nodes of the NeSC training team cluster. The node you should use is specified by the below table:

i. tc03.nesc.ed.ac.uk : user01 - user06
ii. tc05.nesc.ed.ac.uk : user08 - user12

This tutorial involves some editing of files. To edit a file use the command

```
pico <filename> &
```

where <filename> is the name of the file you wish to edit (though you may also use the "vi" editor if you would prefer to).

If your laptop is able to handle X11 windows then you may use a graphical editor, for example:

```
kwrite <filename> &
```

The commands in this tutorial should be run on the same node of the NeSC Training cluster as you have used in the previous tutorial

1. The first step is to just access a data service and discover what data resources are available. If this is a new terminal session from the previous tutorial you will need to run the command

   ```
   module load java ogsadai_wsrf
   ```

2. The java code is already in your account and just needs compiling. If you wish you can inspect the code with the command

   ```
   cat ogsadai_client/LocatingADataService.java
   ```

   Compile this code by typing

   ```
   javac ogsadai_client/LocatingADataService.java
   ```

   and then run the program with the command

   ```
   java ogsadai_client.LocatingADataService
   ```

   You should get the output

```
Locating data service with the following handle:
  http://tc03:8080/wsrf/services/ogsadai/DataService
The following resources are available at this service:
Challenge1
LittleBlackBook
Scratch0
```

Optional: If you are experienced with java edit the code to view the resources available on tc05 and tc07 by replacing the string "tc03" in the code with "tc05" and "tc07".

3. The next stage is to access the LittleBlackBook database and display several entries. The java code for running a simple sql query is already in your account in the file "ogsadai_client/SimpleSQLQueryClient.java". A helper class called "Utilities" that provides some formatting functions, has also been provided (and already compiled). Compile and run the code. You should get the below output

```
Fetch DataService on http://tc03:8080/wsrf/services/ogsadai/DataService for resource
LittleBlackBook
Performing SQL query select * from littleblackbook where id<10
Response:

  1 Ally Antonioletti 826 Hume Crescent, Southampton 01670061244
  2 Amy Atkinson 583 Atkinson Drive, Southampton 06312054624
  3 Andrew Borley 354 Jackson Road, Edinburgh 01057075166
  4 Charaka Chue Hong 750 Pearson Crescent, Southampton 09945916393
  5 Dave Hardman 079 Borley Gardens, Winchester 06725558505
  6 George Hicken 398 Magowan Street, Winchester 09066873297
  7 James Hume 801 Laws Gardens, Edinburgh 00246566355
  8 Malcolm Jackson 743 Krause Lane, Edinburgh 04333719273
  9 Mario Krause 026 Atkinson Gardens, Winchester 01922093483
```

4. The previous code displayed the results already formatted. These results were transferred to your code in the form of a "Response Document". This document describes the status of execution of the task you requested of OGSA-DAI and may also contain result data, as was the case in the previous code. When this document contains result data it also contains information describing the format of the data.
   Modify the code used in the previous section to display the received Response document. This can be done by inserting the line

   System.out.println(response.getAsString());

   between the lines

   System.out.println("Response: ");

   and

   ResultSet rs = rowset.getResultSet();

   Recompile the code and run it again.You should get a result similar to the below output (note that due to the length of the output, parts have been cut out).

```
Fetch DataService on http://tc03:8080/wsrf/services/ogsadai/DataService for resource
LittleBlackBook
Performing SQL query select * from littleblackbook where id<10
Response:
<?xml version="1.0" encoding="UTF-8"?>
<ns1:response xmlns:ns1="http://ogsadai.org.uk/namespaces/2005/10/types">
    <ns1:session id="session-ogsadai-1098d2e1864"/>
    <ns1:request status="COMPLETED"/>
    <ns1:result name="SQLQuery-ogsadai-109a2b3a400" status="COMPLETED"/>
    <ns1:result name="WebRowSet-ogsadai-109a2b3a402" status="COMPLETED"/>
    <ns1:result name="ogsadai-109a2b3a403" status="COMPLETED"><![CDATA[<webRowSet
xmlns="http://java.sun.com/xml/ns/jdbc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/jdbc http://java.sun.com/xml/ns/jdbc/
webrowset.xsd">
<properties>
```

```
<command></command>
<concurrency>1007</concurrency>
...
<columnValue>1</columnValue>
<columnValue>Ally Antonioletti</columnValue>
<columnValue>826 Hume Crescent, Southampton</columnValue>
<columnValue>01670061244</columnValue>
</currentRow><currentRow>
<columnValue>2</columnValue>
<columnValue>Amy Atkinson</columnValue>
<columnValue>583 Atkinson Drive, Southampton</columnValue>
<columnValue>06312054624</columnValue>
</currentRow><currentRow>
...
```

5. The next section of the tutorial looks at more detail into the tasks that OGSA-DAI can achieve. The terminology used in OGSA-DAI for a task is "Activity". In the previous examples a simple activity, in this case an SQLQuery using a WebRowSet to access the data, was constructed, sent to the OGSA-DAI server via a "perform document" (not looked at in this tutorial) and the results received as a response document. Before continuing examine the code to see how these activities were combined and performed.

   An activity in OGSA-DAI can do more than one task at the server end before returning the response to the client. Thus it is possible to construct a workflow of activities acting on the data. The code used this time will perform an sql query, compress the results and finally transfer the compressed results to an ftp site. ***You must modify the code so that the ftp site used is the cluster node you are using.*** This is to avoid problems with security when transferring to a locally running anonymous server.

   After making the edit, compile and run the file ogsadai_client/MultipleActivityClient.java. You should get output similar to the following:

```
Fetch DataService on http://tc03:8080/wsrf/services/ogsadai/DataService for resource
LittleBlackBook
Performing SQL query select * from littleblackbook where id='1'
Response:
<?xml version="1.0" encoding="UTF-8"?>
<ns1:response xmlns:ns1="http://ogsadai.org.uk/namespaces/2005/10/types">
    <ns1:session id="session-ogsadai-1098d2e1866"/>
    <ns1:request status="COMPLETED"/>
    <ns1:result name="SQLQuery-ogsadai-109a2dc7471" status="COMPLETED"/>
    <ns1:result name="WebRowSet-ogsadai-109a2dc7473" status="COMPLETED"/>
    <ns1:result name="GZIPCompression-ogsadai-109a2dc7475" status="COMPLETED"/>
    <ns1:result name="DeliverToURL-ogsadai-109a2dc7478" status="COMPLETED"/>
    <ns1:result name="ogsadai-109a2dc7477" status="COMPLETED"><![CDATA
[<gzipCompressionMetadata>
  <checksum type="adler32" value="3342833185"/>
</gzipCompressionMetadata>
]]></ns1:result>
</ns1:response>
```

6. To see the output, first run the command

   ```
   mv /var/ftp/incoming/<username>.gz ~/
   ```

   (If you don't move the file, the next time you run the code you will get a failure message because the anonymous ftp server default is set up to prevent over-writing of files).

   You can now view the contents of the response using the command

   ```
   zcat ~/<username>.gz
   ```

7. Moving towards the end of this stage of the tutorial, the next exercise demonstrates how a series of activities on one OGSA-DAI exposed resource can exchange data with another resource. These resources may or may not be running on the same OGSA-DAI server. This time the code will use an output datastream to push a series of entries from the LittleBlackBook table to a temporary table in one of the scratch databases. The actvity "SQLBulkLoad" is used for puting the data into the temporary table since this is much more efficient than handling each entry individually. It should be noted that the construction of an output datastream requires the explicit use of sessions which are not covered in

this tutorial. To keep the code simple much of the functionality from the previous examples has been incorporated into the class "DataSource" which is already compiled for you.

To avoid creating a table name that conflicts with other trainees and to share the usage across the OGSA-DAI servers please follow the below two rules when creating your temporary tables:

i. For users user01-user06 it should be contained within the Scratch0 database whilst for users user07-user12 it should be Scratch1.
ii. Use a temporary table name of the format "<username>tmp<some letter(s)>" where <username> is your local username.

Before you can run ***the code it must be modified so that it talks to the scratch resource as just detailed and change the name of the temporary table***. This involves modifying the hostname, the resource name and the table name.

Compile the java file "ogsadai_client/BulkLoad.java" and run the class to get output similar to the below output:

```
Fetch DataService on http://tc03:8080/wsrf/services/ogsadai/DataService for resource
LittleBlackBook
Connecting to DataService for temporary storage
Create table user00
Perform bulk-load from LittleBlackBook to temporary table
Performing SQL query: select * from user00

 Ally Antonioletti       826 Hume Crescent, Southampton
 Amy Atkinson    583 Atkinson Drive, Southampton
 Andrew Borley   354 Jackson Road, Edinburgh
 Charaka Chue Hong       750 Pearson Crescent, Southampton
 Dave Hardman    079 Borley Gardens, Winchester
 George Hicken   398 Magowan Street, Winchester
 James Hume      801 Laws Gardens, Edinburgh
 Malcolm Jackson         743 Krause Lane, Edinburgh
 Mario Krause    026 Atkinson Gardens, Winchester

Drop temporary table
```

8. To conclude this section of the tutorial a join between seperate date resources is performed. This requires an additional data resource which is exposed via the data service on "tc05" and is called "StaffList".

   Optional: If you are familiar with java edit the code from the previous step to access the resource "StaffList" under the data service "http://tc05:8080/wsrf/services/ogsadai/DataService" to bulk load all entries resulting from the search "select * from stafflist where id=1".

9. To perform a join it is neccessary for both sets of data to be joined to be temporarily present in the same database, in this case your allocated Scratch resource. Modify the java file "ogsadai/JoinClient.java" in the same way as for the file "ogsadai_client/BulkLoad.java". Run this class to run the join on "LittleBlackBook" and "StaffList" that searches for the same name in both databases. You should get the output

```
Fetch DataService on http://tc03:8080/wsrf/services/ogsadai/DataService for resource
LittleBlackBook
Fetch DataService on http://tc05:8080/wsrf/services/ogsadai/DataService for resource
StaffList
Fetch DataService on http://tc07:8080/wsrf/services/ogsadai/DataService for resource
Scratch2
Create table user00a with fields name varchar(64), address varchar(64)
Fetch DataService on http://tc07:8080/wsrf/services/ogsadai/DataService for resource
Scratch2
Create table user00b with fields name varchar(64), staffid varchar(20)
Perform bulk-load from LittleBlackBook to Scratch2.user00a
Perform bulk-load from StaffList to Scratch2.user00b
Performing SQL query: select a.name, a.address, b.staffid from user00a as a, user00b
as b where a.name=b.name

 Mike Mineter    12 Near Grantham Street         MM=007

Drop temporary table user00a
```

```
Drop temporary table user00b
```

The OGSA-DAI Grand Challenge

The aim of this part of the tutorial is for each user to carry out a search across 3 databases for a identification code. This code is unique for each user and can be used to construct a url that points to a jpeg image file. This image will be one part of a larger picture. Once you have found your identification code, this will be inserted into a final database so that it can be detected by the program that is building the final picture. This picture will be displayed by projector during the tutorial and is updated regularly to include the parts of the final picture that have been found so far.

The challenge is solved by a progressive translation of a pair of strings to yield your identification code. The three databases/ resources used are Challenge1, Challenge2 and Challenge3. These databases contain table called "challenge1", "challenge2" and "challenge3" respectively and share the common schema:

Column	Name	Type	Length
1	oldvalue	varchar	32
2	identifier	varchar	32
3	newvalue	varchar	32

1. The first step is to generate your starting strings using the commands

   ```
   odgc_encode.pl <username>
   odgc_encode.pl challenge
   ```

   For the purposes of the below algorithm these values will be labelled A0 and B0 respectively.
2. Edit the file ogsadai_client/GrandChallenge.java according to the following:
   a. Replace <username> with your username in the line

      scratchTablePrefix = "<username>";

3. Replace <hostname> and <ResourceID> with the hostname and resource id of your allocated Scratch resource respectively in the lines

   TmpSource tmpSource = new TmpSource(
   "http://<hostname>:8080/wsrf/services/ogsadai/DataService",
   "<ResourceID>");

4. Replace <A0> AND <B0> with your encoded username and the encrypted version of the word "challenge" respectively in the lines

   String A0 = "<A0>";
   String B0 = "<B0>";

- This code will perform the following:

i. first perform a join over resources Challenge1 and Challenge2 using the following steps:
   a. Get all values in Challenge1 where oldvalue = A0
   b. Get all values in Challenge2 where oldvalue = B0
   c. Search both of the above two result sets for a line with the same value of identifier.
   d. Get the corresponding newvalue from Challenge1. This value is called A1.

ii. Next step c is repeated with A0 and B0 swapped over so as to again retrieve a newvalue from Challenge1. This value is B1.

iii. Repeat steps c and d this time starting with the values A1 and B1 and using the resources Challenge2 and Challenge3. The newvalues retreived from Challenge2 are called A2 and B2.

iv. Again repeat steps c and d this time starting with the values A2 and B2 and using the resources Challenge3 and Challenge1. The newvalues retreived from Challenge3 are called A3 and B3.

- One of A3 or B3 is your individual code. Point your web-browser at **http://tc05.nesc.ed.ac.uk/ogsadai/<A3>.jpg** and then **http://tc05.nesc.ed.ac.uk/ogsadai/<B3>.jpg** The wrong answer will be an image just containing the same value as its name. Note some of the correct images may be completely white so do check for this.
- Once you have found your correct individual code it is neccessary to insert this value into the table "challenge4" under the Challenge4 Resource so that the program building the final image can find your part of the image. This is done by editing the file "ogsadai_client/InsertClient.java" to modify the lines

String username = "<username>";

String imageId = "<imageId>";

replacing <username> with your username and <imageId> with either <A3> or <B3> depending on which is your answer. Compile and run this code to insert your value into the database.

- Once all users have found their individual codes the final picture should be complete on the projector.

Where Now?

If you have time left at the end of the practical then you are encouraged to explore the documentation on the OGSA-DAI web site that may be found at http://www.ogsadai.org/documentation/ogsadai-wsrf-2.1/doc/.