SIGCSE 2010 - The 41st ACM Technical Symposium on Computer Science Education
March 10-13, 2010, Milwaukee, WI, USA

Workshop 20: Teaching a Hands-on Undergraduate Grid Computing Course

Session 3: GridNexus Workflow Editor

Feb 27, 2010

Note: *This session is based upon "GridNexus Job Submission" by C. Ferner and B. Wilkinson, February 5, 2010, http://www.csc.uncw.edu/~cferner/ITCS4146S10/assign3S10.pdf,and and Using GridNexus to Create Workflows that use Web and Grid Services" by C. Ferner and B. Wilkinson, December 22. 2009, http://www.csc.uncw.edu/~cferner/ITCS4146S10/assign6S10.pdf*

I. Overview

The purpose of this session use GridNexus to execute programs and grid services on grid servers. We will use the same application of computing the volume and cost of mulch to cover the flower bed as you did in the previous sessions. This time, we will use GridNexus to execute those same applications and move the data between the servers. We will also use GridNexus to make use of Grid services that accomplish the same thing.

II. Task I

Part 1: Downloading and Setting Up GridNexus

1) First, you need to have java version 1.6 or greater on the local machine. If you don't have it, then you will need to install that first from http://java.sun.com
2) Download GridNexus2.02 from www.gridnexus.org. Use the self-extracting file (GridNexus2_02.exe) and install the software.
3) Creating a proxy
   a) Choose Start->All Program->GridNexus->Utils->Proxy->grid-proxy-info. This will report that the program is unable to load the proxy file, but it will also give you the full pathname of where it expects the proxy to be. Write down this pathname.
   b) Choose Start->All Programs->GridNexus->Utils->Proxy->MyProxyLogon. Provide the username and passphrase provided to you. The hostname is coit-grid02.uncc.edu. The Output should be the full pathname you wrote down from step 4a above. Also check the option that says "Write trust roots to ... certificates".
   c) Press the Logon button. You should see "Credentials written to <pathname>" in the message window. Close the MyProxyLogon window.
   d) Choose Start->All Programs->GridNexus->Utils->Proxy->grid-proxy-info. This time you should see the credential subject.
   e) Press any key to close the window

Part 2: Using GridExec

1) In this step, you will create a workflow that uses GridExec to run the two programs you wrote for the first session on two machines and transfer the data files. You will want to have

a login session to both machines to verify the workflow works correctly as well as to clean up. Delete any temporary files (like stdout, stderr, and area_output) on both machines coit-grid01 and torvalds. You will put this workflow together in parts, testing each step. The final workflow will look something like Figure 3.

2) Start GridNexus (Start->All Programs->GridNexus->GridNexus). Then Choose File->New->Workflow.
3) Create a new workflow that runs your first program on a UNCC server. Use the GridExec module (Module Library->Transformers->Grid->GridExec). Create constant boxes that provide: “coit-grid01.uncc.edu:8440” as the Factory Contact, the full path for the Java runtime environment as the Simple Command, the name of your “.class” java program (without the .class extension) as the first argument, then strings with the values for *a*, *b*, and *n* (the number of trapezoids*)* as the next arguments. All of the values need to be given as strings (inside double quotes) including the integer values for *a*, *b*, and *n*. Also remember that the order in which constants are connected to a multi-port is important.
4) Add a Prog module (Module Library->Transformers->Basic->primitives->Prog) and connect the GridExec to it. Add a JXPLDisplay module (Module Library->Sinks->JXPLDisplay) and connect the Prog to it. The workflow should look something like Figure 1.
5) Save it, run it, and verify that the result appears in the file stdout in your home directory on coit-grid01. Then remove the stdout and stderr files.

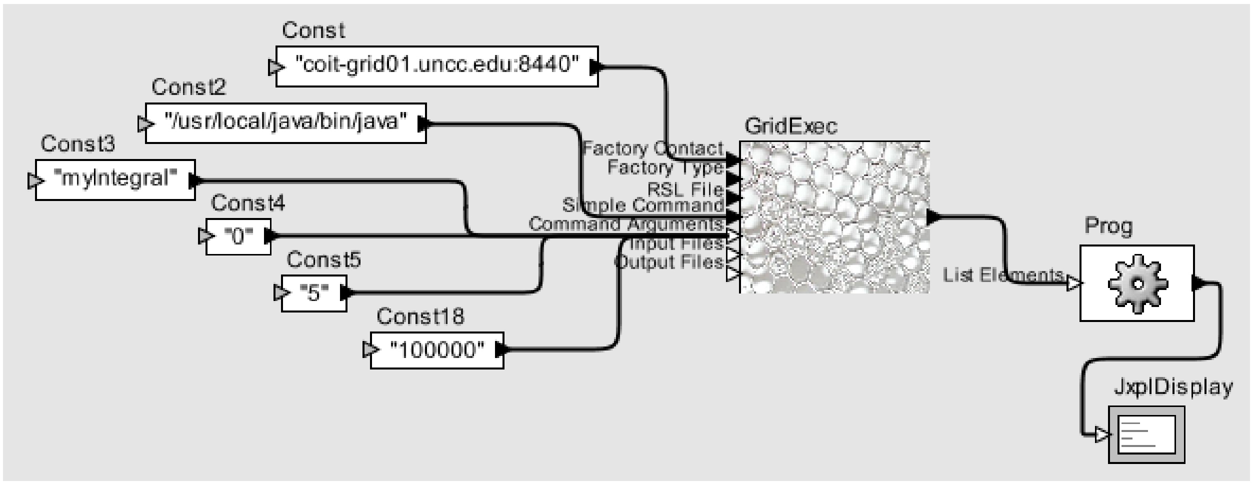
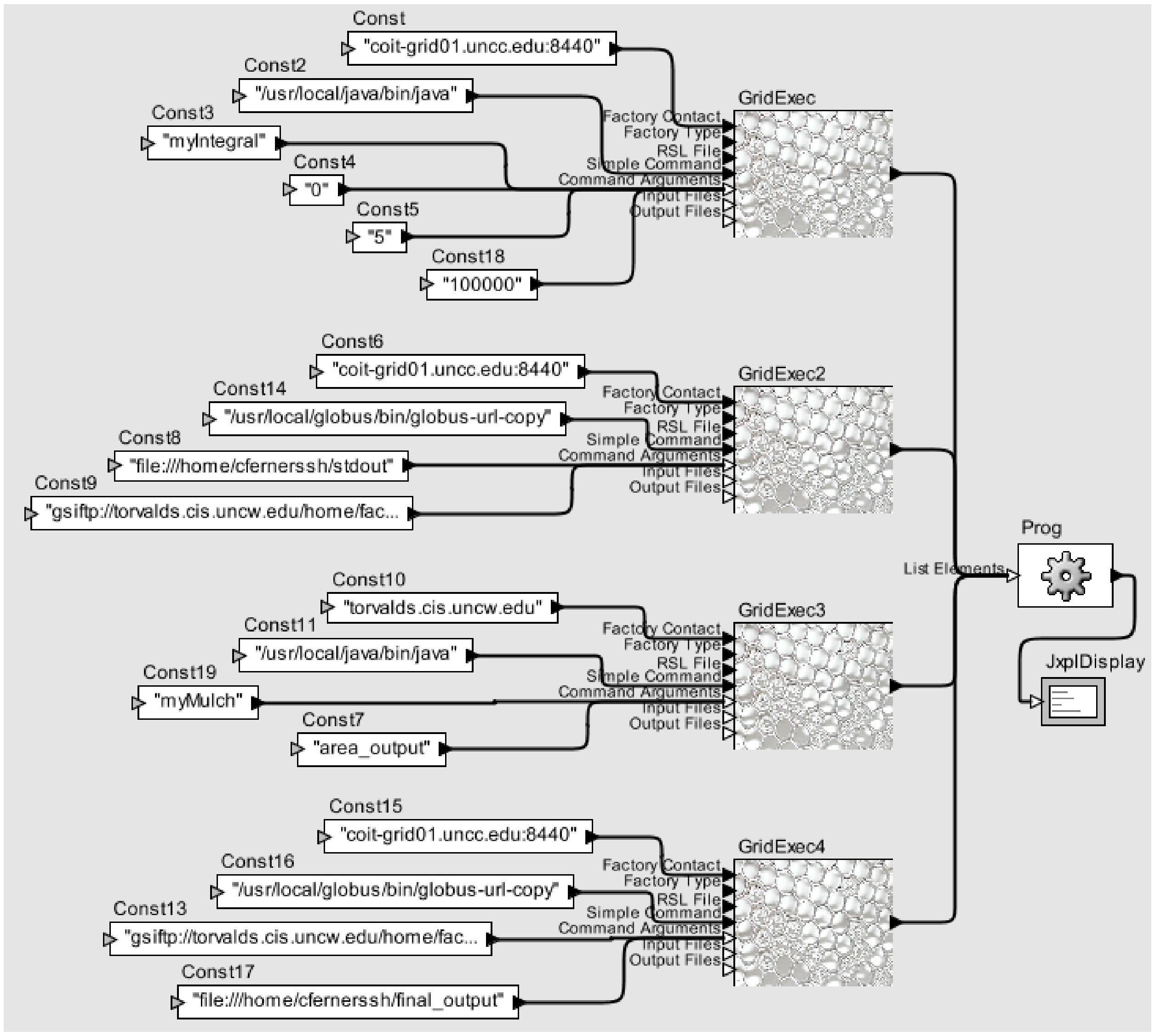


Figure 1: Sample Workflow for One Job

6) You now need to get the output from Step 3 moved to torvalds.cis.uncw.edu in order to run the second java program on it. We can use GridExec to run globus-url-copy to transfer the file to torvalds. Add another GridExec to your workflow to run that command on coit-grid03. The full path of the executable is “/usr/local/globus/bin/globus-url-copy”. The arguments are “file:///<full pathname of your outputfile>” and “gsiftp://torvalds.cis.uncw.edu/<full pathname of where to put the file>/area_output”. Note that since this command is running on coit-grid01, the input file is local. Also note that the pathname to your home directories on the two machines will be different. Your home directory on coit-grid01 is in /nfs-home/ and your home directory on torvalds is in “/home/grid/”. You can run the command **pwd** on both machines to see what the full path to your home directory is.
7) Save it, run it, and verify that the result file is transferred to the proper location on torvalds. Then remove the stdout, stderr, and area_output files on both machines.

8) Add another GridExec to run the second java program on torvalds. The Factory Contact is "torvalds.cis.uncw.edu" (the port is not needed). The command is once again the path to java. The arguments will be the name of your class for the java program and the full path of the file you transferred in Step 6. All of the values need to be given as strings. Also remember that the order in which constants are connected to a multi-port is important. The output of this GridExec should be connected to the Prog Module.
9) Save it, run it, and verify that the result file is transferred to the proper location on torvalds and the correct output is created. Then remove the stdout, stderr, area_output, and another other data files on both machines.
10) The last thing to do is transfer the output file back to coit-grid01. Add another GridExec to your workflow to run globus-url-copy on coit-grid01 to transfer the file back renaming it to final_output. Save it, run it, and verify that the result appears in the file final_output in your home directory on coit-grid01. The final result should look something like figure 2.


Const
"coit-grid01.uncc.edu:8440"
Const2
"/usr/local/java/bin/java"
Const3
"myIntegral"
Const4
"0"
Const5
"5"
Const18
"100000"
GridExec
Factory Contact
Factory Type
RSL File
Simple Command
Command Arguments
Input Files
Output Files
Const6
"coit-grid01.uncc.edu:8440"
Const14
"/usr/local/globus/bin/globus-url-copy"
Const8
"file:///home/cfernerssh/stdout"
Const9
"gsiftp://torvalds.cis.uncw.edu/home/fac...
GridExec2
Prog
List Elements
Const10
"torvalds.cis.uncw.edu"
Const11
"/usr/local/java/bin/java"
Const19
"myMulch"
Const7
"area_output"
GridExec3
JxplDisplay
Const15
"coit-grid01.uncc.edu:8440"
Const16
"/usr/local/globus/bin/globus-url-copy"
Const13
"gsiftp://torvalds.cis.uncw.edu/home/fac...
Const17
"file:///home/cfernerssh/final_output"
GridExec4
Figure 2: Sample Workflow for All of Step2

Part 3: Using WSRF Client

1) In this step, you will create a workflow that uses WSRF to run the two grid services that perform the two parts of the flowerbed-mulch problem. The grid services have already been created and deployed on the servers. You will be created the workflow that uses them. First you need the jar files from the server.
2) Download the jar files from the workshop web page (http://www.csc.uncw.edu/~cferner/SIGCSEWorkshop/) to your machine. Place them in C:\Program Files\GridNexus2.02\lib. The files you need are:
   a) org_globus_examples_services_core_second.jar
   b) org_globus_examples_services_core_second_stubs.jar
   c) org_globus_examples_services_core_third.jar
   d) org_globus_examples_services_core_third_stubs.jar
3) You will need to exit out of GridNexus and restart it for these changes to take effect. Create a new workflow. Drag a WSRF Client (Module Lirary->Transformers->Grid->WSRF Client) to the workflow. Right click on it and choose "Configure". Provide the following information:
   a) Factory URL: "https://coit-grid01.uncc.edu:8440/wsrf/services/examples/core/second/MathService"
   b) Addressing Locator Class: "org.globus.examples.stubs.MathService_instance_second.service.MathServiceAddressingLocator"
4) If the addressing locator class was entered correct, the WSRF Client will now be populated with input ports for the operations the service provides. It should like something like Figure 3. If it doesn't then you probably made a typing mistake.

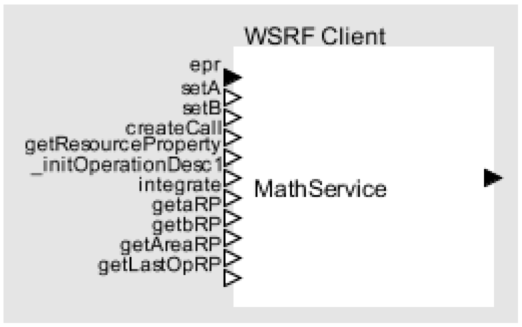


Figure 3: After configuring the WSRF Client

5) Create a workflow that sets *a* and *b*, and then calls integrate (giving the value of *n*). The values should be integers (not strings) this time. Then copy and paste the WSRF Client to get the area. Connect a constant box with the value of "void" (with quotes) to the getAreaRP input. Connect these to a Prog which should then be connected to a JXPLDisplay. The workflow should look something like Figure 4. Save the workflow and run it. You should get a JXPL Display window with the correct area.

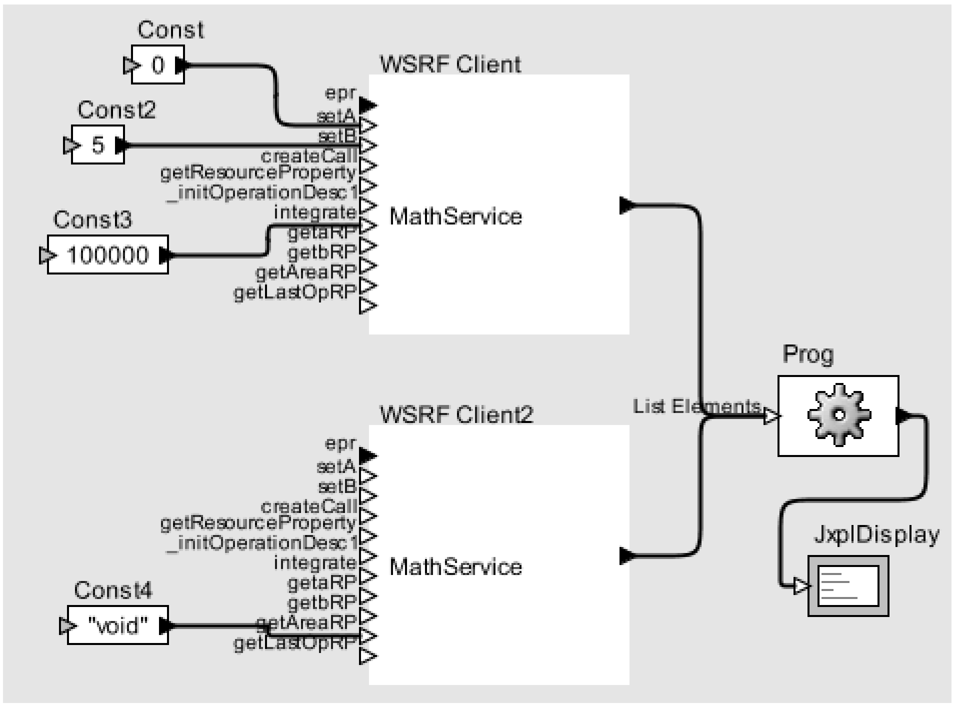


Figure 4: Sample calling the Integrate Grid Service

6) Add another WSRF Client to the workflow. Configure it with the following information:
   a) Factory URL: "https://torvalds.cis.uncw.edu/wsrf/services/examples/core/third/MathService
   b) Addressing Locator Class: "org.globus.examples.stubs.MathService_instance_third.service.MathServiceAddressingLocator"
7) Set the thickness to .333333 feet and cost to $2.90. Connect this WSRF Client to the Prog.
8) Copy and paste this new WSRF Client to make a new one. Disconnect the output of the WSRF client that returns the area from the Prog. Instead, connect it to the "mulch" of this new client. Also provide a "void" to the getCostRP operation to find out the total cost. Finally, connect this newest client to the Prog. The workflow should look something like Figure 5.
9) Save the workflow and run it. You should get a JXPL Display window with the final result. Try conneting the "void" constant to the getVolumeRP port. Also try changing some of the other parameters to see what effect they have.

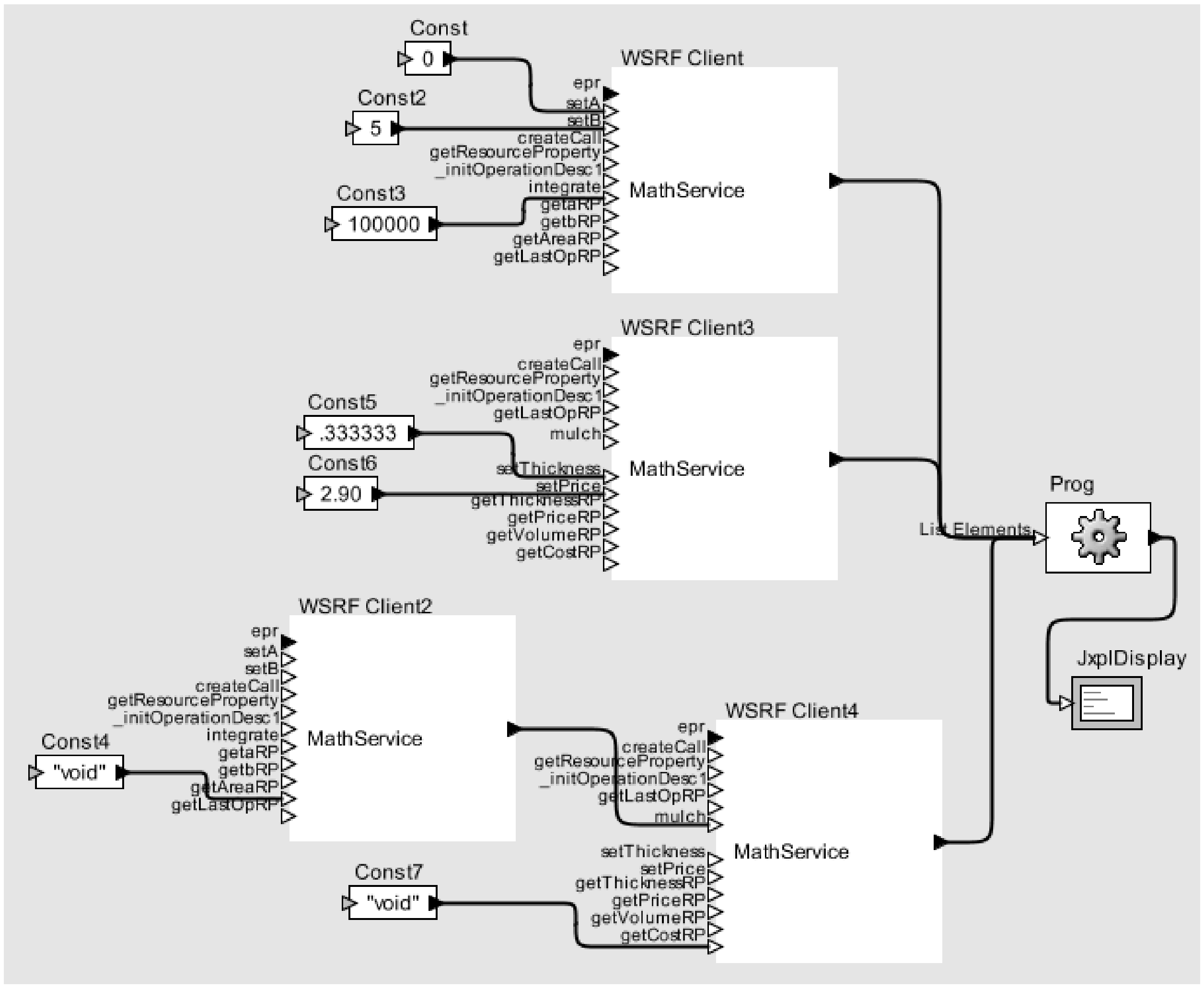


Figure 5: Final Workflow