Toward a Graphical User Interface for Grid Services

Michael Wood Clayton Ferner Jeff Brown
Department of Computer Science
University of North Carolina at Wilmington
Wilmington, NC 28403
{mtw9788,cferner,brownj}@uncw.edu

Abstract

The goal of the UNCW Grid project is to produce a user-friendly graphical environment for assembling and executing Grid applications. In this paper, we provide examples of the work done for this ongoing project. In particular we demonstrate a "drag-and-drop" approach to assembling complex Grid applications from individual Grid services. We also show how our graphical environment interacts with existing services to install and execute user applications on remote machines.

1. Introduction

Grid services are one of the most intriguing areas in the computer science field and are likely to have a significant impact on the scientific and business arenas in the near future [1]. A Grid is a networked infrastructure that enables collaborative use of distributed resources, using general purpose protocols and interfaces, and operates in an environment that is free of centralized control. The goal of a Grid is to eliminate organizational and geographical boundaries and integrate resource islands so that these resources are available to all users.

Currently, interacting with Grid services can be a tedious and complex process. Presently, a user must be capable of writing Java™ client code that incorporates Open Grid Services Infrastructure (OGSI) [3] and Grid Security Infrastructure (GSI) [3] classes. Specifically, this means creating an interface, generating the Web-Services Description Language (WSDL), generating the client side and server side stubs, implementing the interfaces, creating the deployment descriptor (WSDD), generating the jar and gar files, and finally deploying the service [3].

Furthermore, many demonstrations of Grid services use command line interfaces, and the processing is performed in batch mode. Although this is functional, it appears from the user's point of view to be no different than the computing model used for supercomputing since the 1950s.

As a means of facilitating the adoption of Grid technologies by the scientific community, we have developed an innovative graphical user interface (GUI) to interact with Grid services and applications. A GUI interface to a Grid will become an essential component of the Grid just as the web browser is essential to the World Wide Web. It is very difficult to explain the benefits of URLs and protocols such as HTTP to a user without using a browser. It took the graphical format of web browsers for users to comprehend and visualize the power of the Web.

Our project has three components. The first is a set of Grid services for access to a variety of applications. The second is a new language called JXPL which is designed for scripting Web and Grid services. Finally, there is the GUI for creating JXPL scripts. Although we introduce JXPL, it is out of the scope of this paper.

The paper is organized as follows: section 2 gives background information about the GUI and JXPL along with a simple example; section 3 provides details of the interface between the GUI and a Grid service along with an example; section 4 discusses the creation of new abstractions and functionality; and section 5 gives some concluding comments.

2. Background

2.1 UNCW GUI Implementation

The UNCW GUI is based on Ptolemy [5], an open source project from the University of California at Berkeley. By design, the GUI is intended for users with little or no programming experience. However, the GUI may be used by more advanced users as well. Figure 1 is a screenshot, which we will describe in detail below, of the GUI. It has modules shown as boxes that consist of source code to execute a specific operation along with input and output ports. The input ports allow parameters to be passed into the module. For example, a text box containing a file name can be connected to a module that contains source code to perform a file transfer. An output port can be connected to a display module, such as a text display, or the input port of another module. More

complex applications are created by chaining together several modules.

Working within the GUI environment is fairly easy and straight forward. A frame in the upper-left corner provides convenient access to system tools and additional module libraries. The system tools include objects such as text boxes, HTML output displays, and file I/O modules. The user library modules can be user defined or obtained from other authors. To construct new workflows, a user drags a module from the module library window onto the palette. The modules on the palette can then be connected by clicking on the port of one module and dragging to a port of another module.

2.2 JXPL

To give the modules real functionality, we created a new scripting language called JXPL. JXPL is a LISP inspired language whose scripts are written in XML. DOM elements replace the LISP structures of atom and list, and XML tags supplant the endless parentheses found in LISP S-expressions. Figure 2 shows a simple JXPL example that multiplies 3 and -7/22.

2.3 Google™ Search Example

The first example demonstrates usage of the GUI to perform a Google™ Search and process the results of that search. Specifically, we are interested in finding all of the links (anchors) in the pages that the Google™ search finds. Figure 1 shows the work flow to perform this processing. The first three modules produce a JXPL script. The module labeled *Google Search* contains the string "Globus GT3" which is the search string. The *Google Search* module writes JXPL script to perform the Google™ search on the search string and obtains a list of URLs. The list of URLs is piped into the *HTTP* module. The HTTP module writes script that obtains the actual HTML files for each URL. The *Tag Filter* module writes script that filters the HTML code and outputs only the certain items that have been prescribed through a parameter box. The parameter box can be accessed by right-clicking on the module and selecting "Configure." For this example, the Tag Filter module is configured to return anchors (i.e. returns all <a> tags). The *JXPL Interpreter* module executes the script and returns an XML result. The *XSL* module produces HTML source code according to the style sheet provided by the *Constant Value* box. The output of the XSL module is then saved to disk by the *Tmp File* module. Finally, the results are exhibited in a HTML display window. Figure 3 shows the results of running the work flow described in Figure 1. This example demonstrates the look and feel of the GUI as well as how various modules can be linked together to form one application.

3. Using the GUI with Grid Services

In order to make use of Grid services and applications using the GUI, we need an interface between a Grid service and JXPL. We would like to use the GUI to run a variety of applications on a variety of systems, including high performance and parallel systems. Our approach is to develop a generic Grid service which executes on remote machines. This service needs to be capable of executing any desired application as well as a few basic operations such as a file transfer. The Generic Grid Service is a Java program that is deployed as a service on a remote machine. The API of the service is shown in Figure 4.

A client Java program can be easily written and run on the local machine to interact with the Generic Grid Service. The client can start an application and then interact with it through calls to the read() and write() methods. The client can transfer files to the remote machine by using the setOutFile(), outFile(), and the fileTransfer() methods. The client can also transfer files from the remote machine by running the Unix "cat" program, using the read() method, and storing the returned String to a local file. Since we want to interact with the Generic Service using the GUI, the JXPL code will be used as the client program.

The Generic Grid Service uses a configuration file to determine what applications are available to the remote user. Requests to run applications not included in this file are denied. In section 4, we discuss a *Registration* service that allows authorized users to add new applications to the configuration file.

3.1 Traveling Salesman Problem Example

In this example, we will use a parallel implementation of a program to solve the Traveling Salesman Problem (TSP) that runs on a cluster of computers. This problem was chosen because it is relatively easy to implement in parallel and makes use of other modules in the GUI.

In Figure 5, eleven zip codes are entered in text boxes and serve as the initial inputs. The *Distances* module creates JXPL script that contacts MapsOnUs.com to get the pair wise distances between each pair of zip codes. The *TSP Client* module writes a script that interacts with the Grid service that runs the parallel TSP program on the cluster. The TSP program finds a circuit of the cities with the shortest total distance and returns that circuit. Then, as in the Google™ Search example, the JXPL interpreter executes the script and returns an XML result. This is converted to HTML according to the style sheet, saved, and then displayed.

Figure 6 shows the results from this example. The style sheet allows the circuit to be displayed in an elegant HTML format. Note that the arrows between the cities

are actually hyperlinks that will return a web page from MapsOnUs.com with step-by-step directions between the cities.

4. Evolving Functionality

There are two features of the GUI that enable the functionality to grow and evolve into more complex tasks. First, the GUI itself allows for a workflow to be enclosed into a single module, thus creating an abstraction. Second, the generic client can be used not only to *run* different applications but to *create* new applications.

4.1 Creating a new abstraction

One can see from the Google™ Search and TSP examples that it is possible for the palette to get congested. The GUI enables users to take a section of a workflow and turn it into a new single module. In Figure 7, we have extracted the JXPL Interpreter, XSL, Tmp File, and HTML Display modules from the previous examples to create a new module containing these components. Figure 8 shows the Google™ Search workflow that is now much cleaner using this new module. Furthermore, this same module can be reused.

4.2 Creating a new application

Figures 9-12 demonstrate how the GUI can be used to create and use a new Grid application. First we needed to create a *Generic Client* module. This module interacts directly with the Generic Grid Service. The client can be controlled by the parameters passed to it, which are the URL of the Grid service, options, command, and data. The command is the name of the application on the remote machine. The data will be written to the service using the write() or fileWrite() methods of the service. The options will modify the behavior of the client. For example, a "-w" option indicates that the client should use the fileWrite() method instead of the write() method so that the data will be saved to a file on the remote machine.

In order to deploy a new application, the user will want to transfer a program to the remote machine and possibly compile it. These two operations have been created using the Generic Client, as shown in Figure 9 and Figure 10. For the file transfer, the parameters given are such that the client will cause the data, which is a local file name, to be read and written to the remote machine. A new module called *File Transfer Client* is created with these components, except that the local and remote file names are the input and output. Another module is created that can compile a program once it has been transferred. Figure 10 shows a new module that uses the Java™ compiler. This time, the command given to the Generic Client is a string concatenation of the command "javac" and the file name, but there is no data. The source program name is the input port of the new module, and the output is the file name of the compiled version.

Another module that we need before we can create and use new applications is a *Registration Client* and its corresponding *Registration Service*. The Registration Service is restricted to *authorized* users. The registration process involves adding a program to the configuration file of the Generic Service, only if that program is not already in the configuration file. The registration process will also check to see if the program is already in the path of execution. If the program is not in the path, then the registration process will move the program to a location where it will be in the path. The input port to the Registration Client is the file name of the executable program, and the output ports are the URL of the Generic Service, the options, and the command. These three output ports are the first three required input ports of the Generic Service.

Figure 11 shows a new module that will create a new application. The input to the module is a local program file name. The new module will transfer the file to the remote machine, compile the program, and register it. Figure 12 shows how we can create a new module to execute the new application once it has been registered. The three outputs from the Create New Application module can be given to the Generic Client so that it will be able to run the new application. These outputs then can be used as part of a new module created specifically to run this new application. The input to the new application module is the data, and the result of the application is the output.

5. Conclusions

This paper introduces the UNCW GUI and the JXPL scripting language. These components can make Grid service interaction easier without taking away from Grid functionality. We have shown two examples of using the GUI and JXPL to perform a refined Google™ search and to solve a traveling salesman problem in a Grid environment. In addition, we have exhibited a way to use the GUI to execute user applications remotely, and the level of abstraction that the GUI provides.

The GUI that we have developed will be tested on the North Carolina Grid. The NC Grid is being built using computing resources from universities in North Carolina. We will be deploying this GUI on a few of these campuses in the next year, and the GUI will be used by scientists doing research in Chemistry, Bioinformatics, and Computer Science. Even though it is a newly created interface, we are confident that the availability of this GUI will facilitate the use of the Grid by these researchers.

6. References

[1] Zhang L. Chung J., Zhou Q., “Developing Grid computing applications, Part1, Retrieved from http://www-106.ibm.com/developerworks/grid/library/gr-grid1, November 2003.

[2] Foster, C. Kesselman, J.M. Nick, S. Tuecke, Physiology of the Grid, Retrieved from http://www.globus.org/research.papers/ogsa.pdf, November 2003.

[3] Borja Sotomayor, The Globus Toolkit 3 Programmer’s Tutorial, Retrieved from http://www.casa-sotomayor.net/gt3-tutorial/, November 2003.

[4] Foster, What is a Grid? A Three Point Checklist, Retrieved from http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf, November 2003.

[5] Edward Lee, Ptolemy, http://ptolemy.eecs.berkeley.edu/ , June 2003.

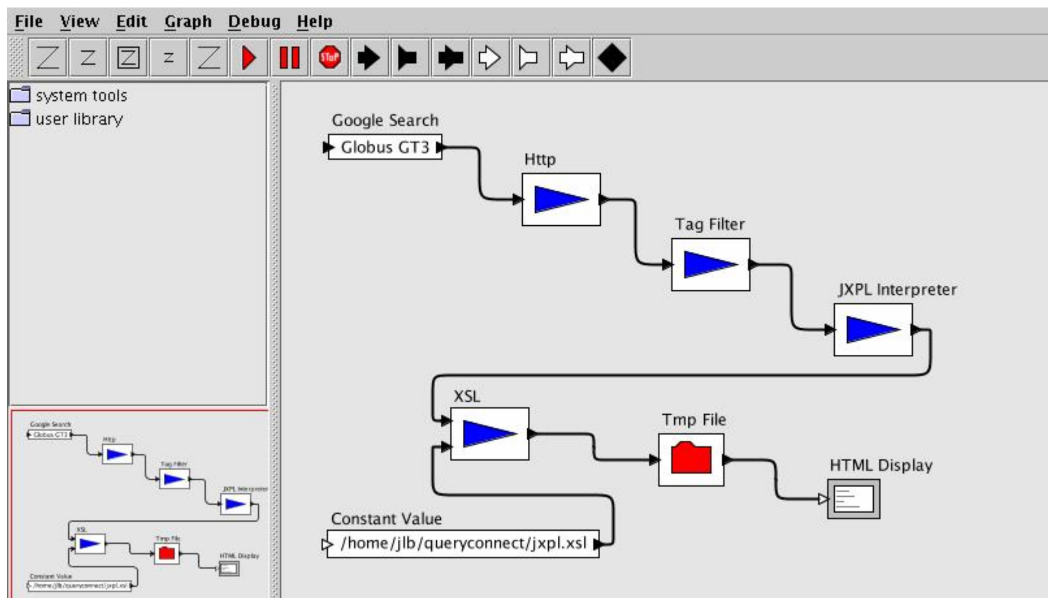

Figure 1: Workflow for the Google™ Search example

```
<?xml version="1.0" encoding="UTF-8"?>
<jxpl:list xmlns:jxpl="http://www.jxpl.org/script">
    <jxpl:primitive name="Arithmetic">
        <jxpl:property name="operation" value="multiply"/>
    </jxpl:primitive>
    <jxpl:integer value="3"/>
    <jxpl:rational numerator="-7" denominator="22"/>
</jxpl:list>
```

Output

```
<jxpl:rational numerator="-21" denominator="22"/>
```

Figure 2: JXPL script that multiplies 2 numbers

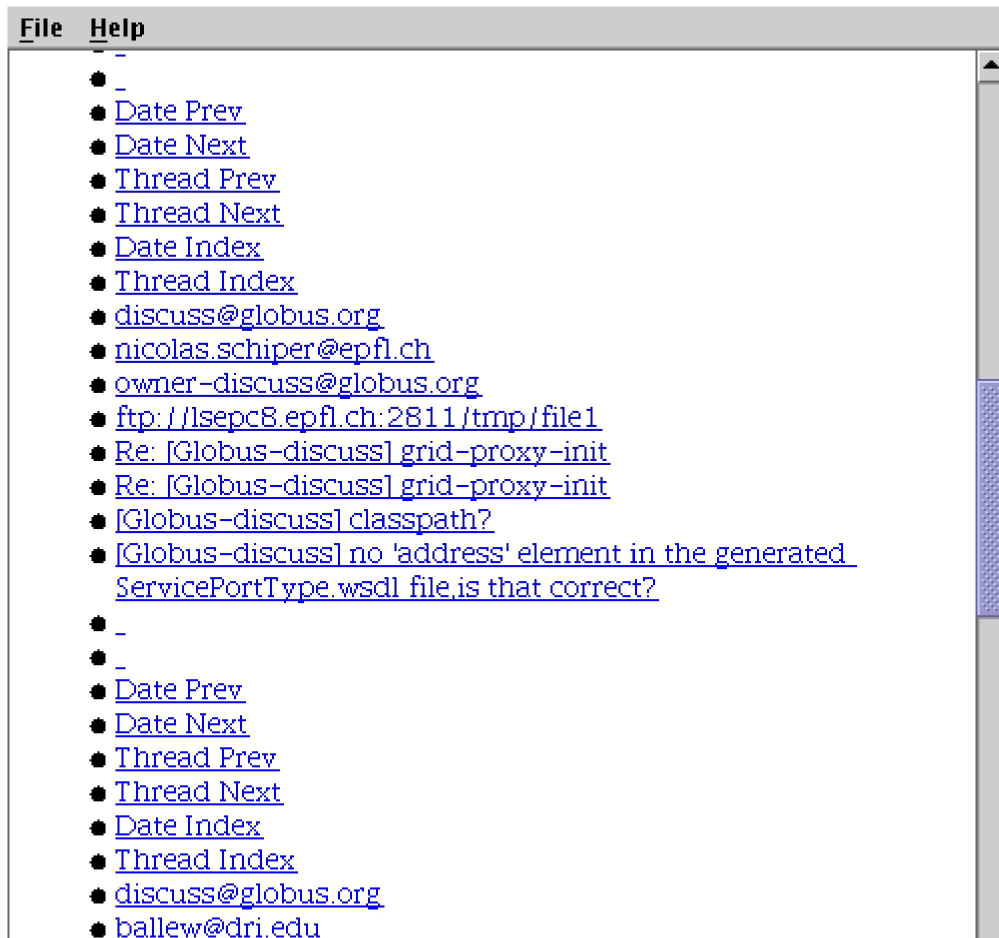

Figure 3: Sample of the results of the Google™ Search example

`String readErr()` – reads from the process' error stream and returns it as a String.
`void readConf()` – reads the configuration file that contains the list of executable applications.
`boolean start(String args)` – creates and begins executing the process.
`void write(byte [] output)` – writes a byte array to the input stream of the process.
`String read()` – reads a byte array off of the process' output stream.
`void stop()` – `terminates` the process and the connection to the service.
`String setOutFile(String filename)` – creates a file on the service side machine.
`void fileWrite(byte [] output)` – writes a byte array to the file created by the setOutFile() method.
`void outputToFile()` – redirects a process' output stream to a file created by the setOutFile() method.

Figure 4: Generic Grid Service API

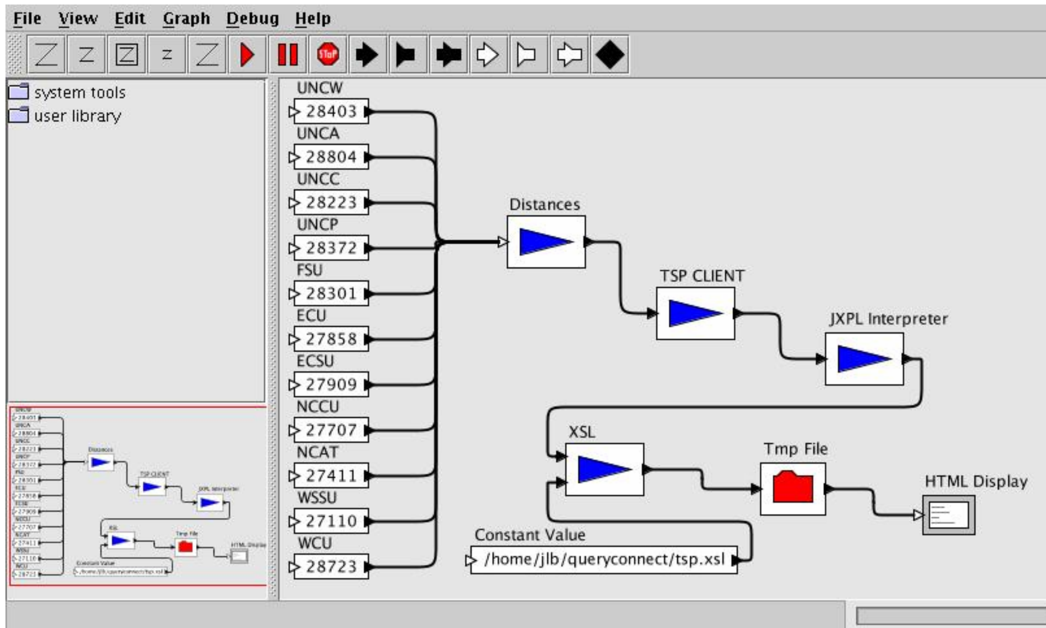

Figure 5: Workflow for the traveling salesman problem example

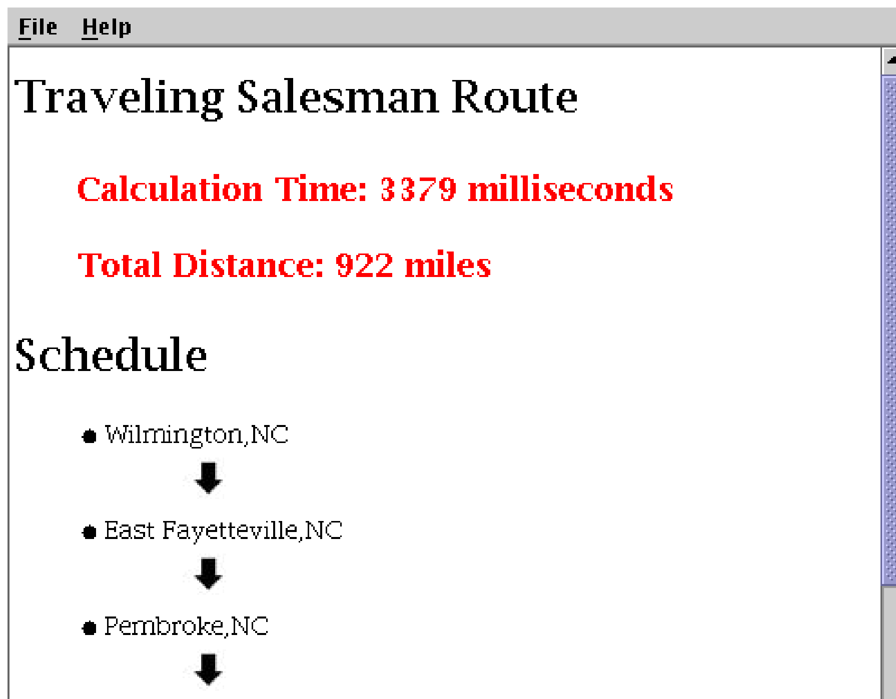

Figure 6: Results of the traveling salesman example

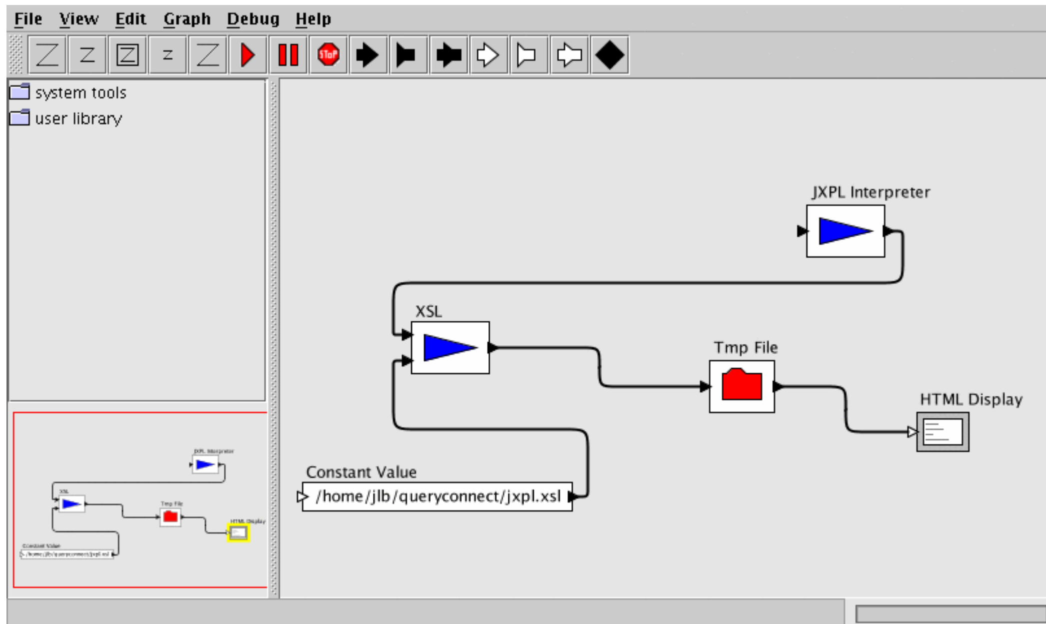

Figure 7: Creating an abstraction of the JXPL Interpreter, XSL, and HTML Display operations

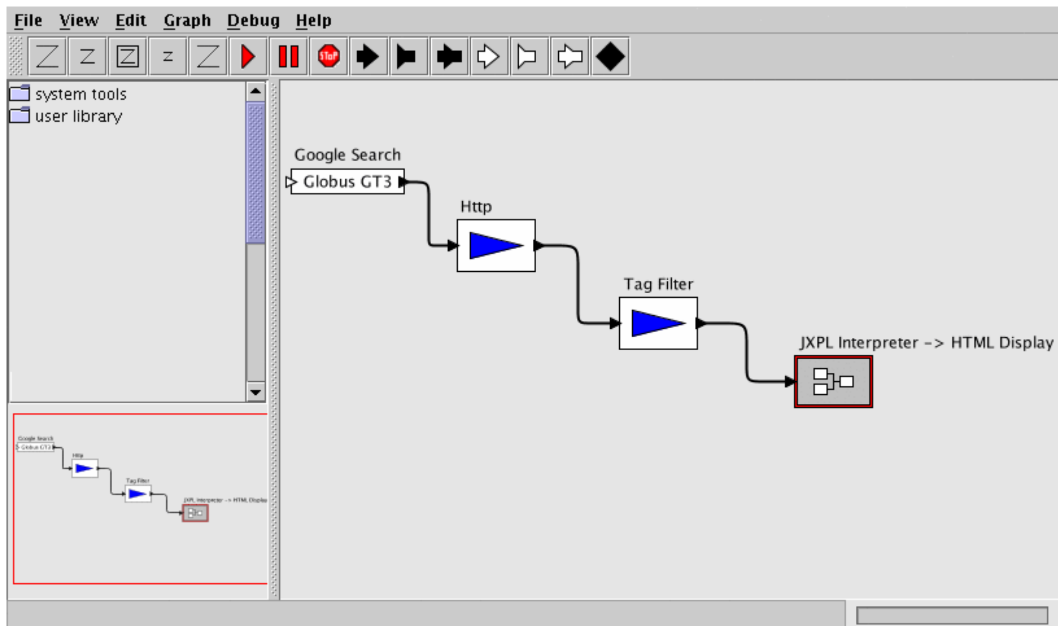

Figure 8: The Google™ search example with the new module that contains the JXPL to HTML Display

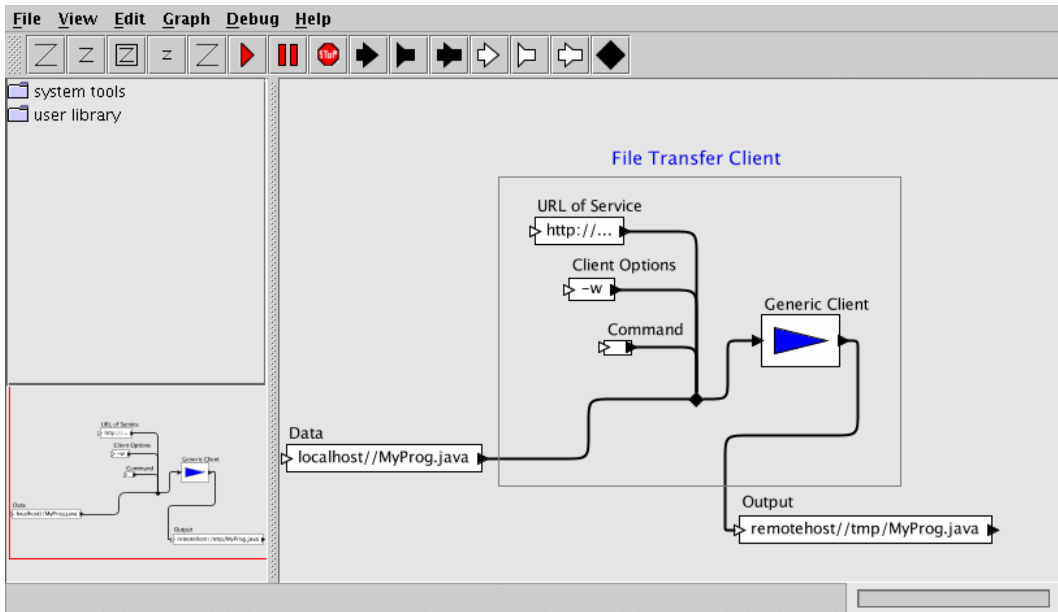

Figure 9: Creation of a File Transfer Client Module

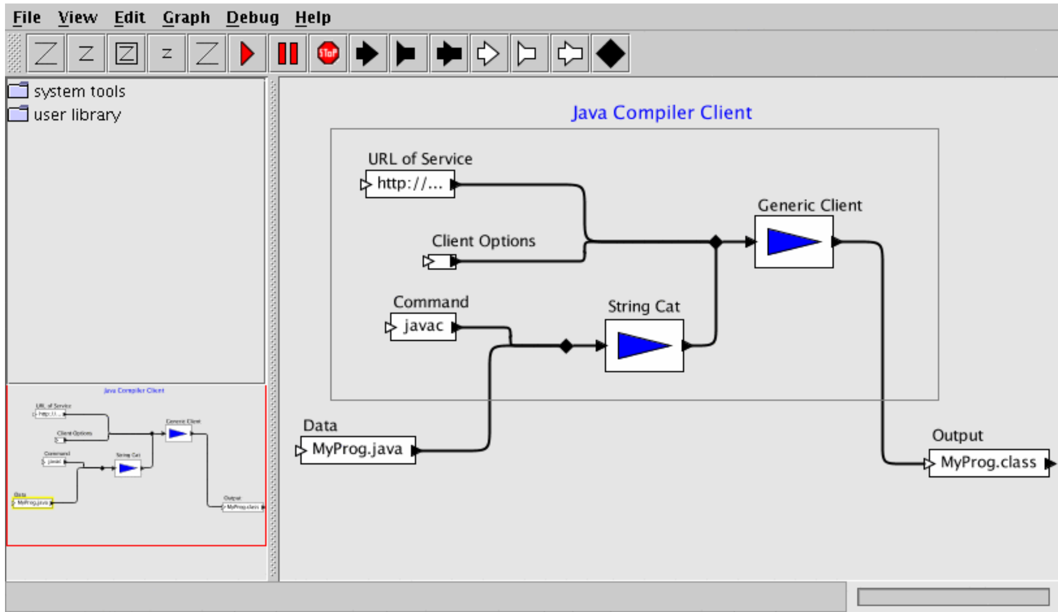

Figure 10: Creation of a Java Compiler Module

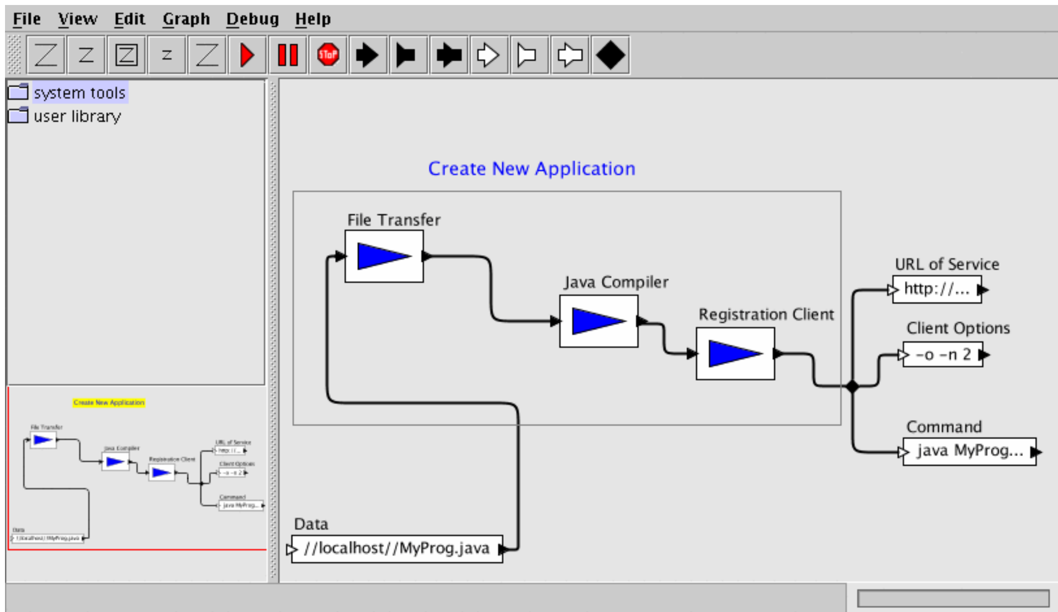

Figure 11: Adding the Register Client component

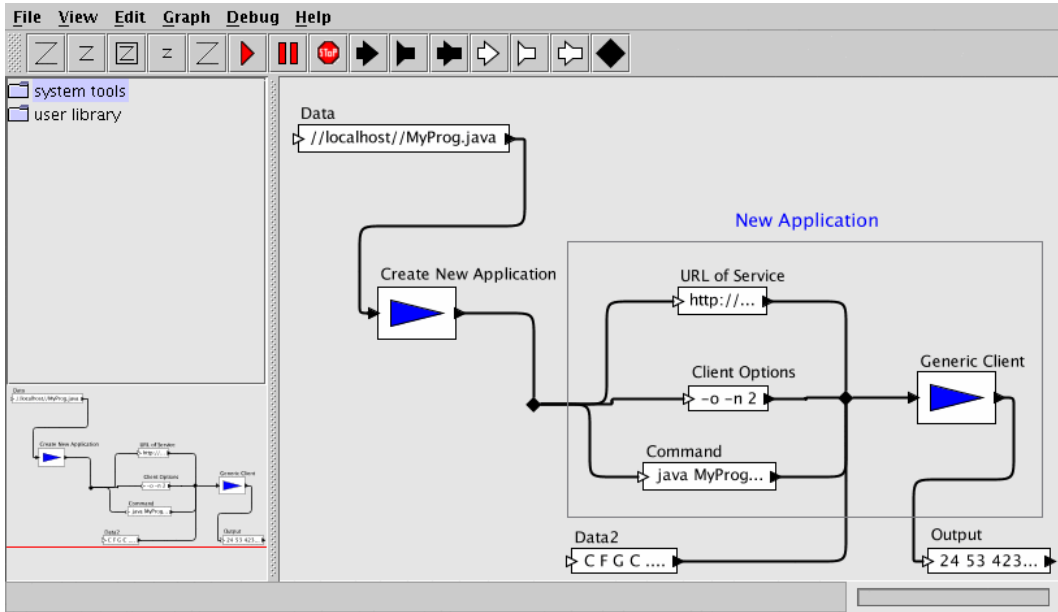

Figure 12: Final workflow to transfer, compile, and execute a Java™ application