GridNexus: A Grid Services Scientific Workflow System

Jeffrey L. Brown, Clayton S. Ferner, Thomas C. Hudson, Ann E. Stapleton, Ronald J. Vetter[a],
Tristan Carland, Andrew Martin, Jerry Martin, Allen Rawls, William J. Shipman, and Michael Wood
University of North Carolina Wilmington, USA

Abstract

We introduce GridNexus, a graphical system for creating and executing scientific workflows in a grid environment[1]. GridNexus allows the user to assemble complex processes involving data retrieval, analysis and visualization by building a directed acyclic graph in a visual environment. Workflows in GridNexus are described by a script written in a language called JXPL. The script can be executed either locally or in a remote managed-job environment. GridNexus enables the composition of larger software modules from smaller ones to build very complex tasks and separates the graphical user interface (GUI) from execution of the workflow. This increases system flexibility while retaining interactive capabilities. We compare our approach with that of similar projects, and provide an assessment of the system by highlighting two real-world scientific applications that have taken advantage of the GridNexus environment.

Keywords: grid computing, workflow programming, bioinformatics, computational chemistry, web and grid services.

1. Introduction

1.1 Overview

In recent years there has been significant research and development related to grids, which promise to provide resource sharing across many domains under decentralized control. The problem of managing grid workflows has received much attention lately, including a workshop at the Global Grid Forum [1]. Implementations of the Open Grid Services Architecture (OGSA) [2] and the emerging Web Service Resource Framework (WSRF) [3] standards have changed the way we work with remote computational resources. In the past, using a remote resource typically involved logging in, sending the appropriate request and receiving the results. Modern grid technology allows seamless authentication and authorization and provides standard interfaces for access to remote services. This makes it much easier to automate the process of accessing and using distributed computing resources.

[a] Contact Author:
Department of Computer Science
UNC Wilmington, Wilmington, NC 28403, USA
vetterr@uncw.edu

[1] A beta version of GridNexus has been released and is available at http://www.gridnexus.org

Today, many demonstrations of grids still use command line and batch mode processing. This is the style of computation used since the first days of computing. It is difficult for a novice user to discern the difference between a supercomputing environment and a grid computing environment. Both require the submission of a job, scheduling and execution of that job, and collecting the results at a later time. For example, suppose a scientist wishes to retrieve a large number of DNA sequences from a database, compare these sequences with those in another database through a BLAST search [4] and view the results. A typical scenario would be for the scientist to log into one machine to perform the query, FTP the results to another machine, log into that machine, perform the BLAST search, and then download the results to the local machine. All of this would probably be done using a command line interface.

If grid technology is to gain widespread acceptance, there must be better interfaces: ones that are more intuitive, graphical, easy to use, but most importantly that hide the complexity of the underlying architecture. Without being able to visualize how a grid will help them, a scientist will likely continue performing their tasks in the same labor intensive way that they have used for years.

Our ultimate goal is to be able to seamlessly integrate disparate high performance computing resources on the North Carolina Research and Education Network into a statewide grid computing environment which does not require scientists to know the underlying network and computational infrastructure.

1.2 Previous Work

We previously developed a flexible architecture for scientific processing that supports a high level of abstraction [5]. After user testing and review of the advances made by other projects of this type, we designed our new grid application builder to incorporate both a highly functional graphical user interface and an extensible, high-level scripting language. We designed our system with execution separated from the interface and with layering so that high-level processes (for scientific users) encompass lower-level processes (used by programmers). The scientist sees a dataflow, with modules carrying out processing steps on the data whereas the programmer creates workflows that enable the building of modules and deals with the actual steps in execution. In addition, an important design criteria was

that modules be re-usable, to avoid wasted effort in script-writing, and that both interactive and batch processes be supported in the workflows. We are convinced that in order for a new system to be widely adopted, it is essential that the user interface be attractive and easy-to-use. We choose to adapt an existing interface that had this characteristic rather than devote substantial resources to a new design.

1.3 Related Work

There are several projects currently under development whose aim is to address a wide variety of grid computing problems [6]. These efforts fall into the following categories: (a) development of a graphical user interface (e.g., Kepler and Taverna), (b) workflow specification (e.g., Pegasus and DAGMan), (c) job submission and scheduling (e.g., GridFlow and CONDOR), (d) resource management and monitoring (e.g., Globus Monitoring and Discovery System), and (e) file management (e.g., GridFTP). The Grid portal framework, whose focus is on providing access to grid resources via a web browser, offers another approach to creating grid computing environments.

Even though there are many different grid projects, we discuss only those projects that most closely relate to GridNexus. This is not intended to be a comprehensive list or complete survey of the field, but rather a representative set of grid projects that directly relate to our work.

1.3.1 Kepler

Since GridNexus and Kepler [7] are both based on Ptolemy II, an open source project from the University of California at Berkeley [8], it is easy to make a direct comparison between these two systems. Consider the problem of creating a new GUI module, or *actor* in the Ptolemy II vernacular. The easiest approach to this task is to extend one of the Ptolemy II classes so that it can take advantage of the dynamic configuration capabilities of Ptolemy II.

In the Kepler setting, the new class is responsible for carrying out the operation that the GUI module represents, such as a database query or a mathematical calculation. Of course, this work might be delegated to another class, but ultimately the new module class is responsible for getting the work done and handling the input and output. In this approach computation occurs in the same code used to create the actor, tying the computation to the user interface.

The functionality of the software modules in GridNexus are provided through an XML based scripting language called JXPL, which is fully described in section 2.2. The modules do not produce the results directly, but rather produce a JXPL script which, when evaluated, produces the result. This separation provides flexibility that will allow GridNexus to operate in more complex environments. The execution of the JXPL script might be performed by a processor running locally, or it could be submitted by the client to be run in a managed job environment. Furthermore, applications other than the GridNexus could be used to create JXPL scripts. With Kepler, execution of the workflow is part of a complex system that includes the GUI.

1.3.2 OGSA-DAI

OGSA-DAI [9] is a project initiated by the UK Database Task Force concerning data access and integration (DAI) in a grid environment. The OGSA-DAI software is included as a contributed part of the Globus Toolkit 3.2. OGSA-DAI provides a system for managing a special type of grid service called a Grid Data Service (GDS). Services that implement the GDS interface work seamlessly together. For example, one GDS could provide database access, while another provides an XSL transformation. OGSA-DAI manages the delivery of data from the database to the transformation service. The result of the transformation could be delivered to another service, a file, GridFTP, etc. In addition to a common data format, there are other important aspects of the exchange that OGSA-DAI handles. In particular, very large result sets are delivered in pieces to avoid taxing machine resources.

The type of workflow management offered by GridNexus is complementary to the specialized data transfer management offered by OGSA-DAI. GridNexus models handle interactions between many different types of processes, some of which could be OGSA-DAI workflows.

1.3.3 Taverna

Taverna [10] is another scientific workflow project similar to Kepler and GridNexus, except that it does not yet support Grid services. Taverna is similar to GridNexus in that it uses an XML-based scripting language to describe the processing that should be executed. The Taverna GUI is used to visualize the workflow but not to build or modify it. Both Kepler and GridNexus use a GUI to create and execute workflows.

The scripting language that Taverna uses, called XSCUFL, is somewhat like Moml, the XML modeling language used by Ptolemy II, although much simpler. XSCUFL contains tags such as: “processor”, “source,” “sink,” and “link.”

In GridNexus, the new module has the relatively easy task of writing JXPL script that will carry out the operation. I/O type is not an issue because the GUI is simply writing script. The execution of the script may be done by a JXPL processor running on the client machine or it could be handled by a processor running as a

persistent Grid service. The JXPL processor could also run under some form of managed job service.

1.3.4 Grid Portals

Grid portals allow users access to high performance computing resources via an easy to use web page interface. Portals provide access to grid technologies through sharable and reusable components for web-based access to scientific and business-oriented applications. Sharable components allow the portal developer to quickly create grid portals from provided libraries that support baseline grid technologies (such as file transfer, job launching an monitoring, and access to information services), freeing the developers to concentrate on the specialized needs of a particular scientific community [11].

A few well known grid portal projects include: (a) the Legion grid portal, (b) GridSphere, (c) GridScape, (d) GridSpeed, and (e) the National Partnership for Advanced Computational Infrastructure (NPACI) Grid Portal Toolkit or GridPort. Each of these projects leverages standard technologies to provide information services that portals can access and incorporate. Although GridNexus is not technically a grid portal, it does address a problem similar to that of grid portals, namely to make grids readily accessible to users. We are currently exploring the integration of GridNexus with the statewide Bioinformatics Portal being created by the University of North Carolina at Chapel Hill.

2. Our Grid Environment

We have created a grid environment that consists of a GUI, called GridNexus, and a scripting language, called JXPL. In our environment GridNexus allows a scientist to create workflows, JXPL specifies the workflows, and a JXPL processor (which can run anywhere, including in the GUI or on a separate server) executes JXPL scripts.

2.1 Graphical User Interface

The graphical user interface of GridNexus is derived from Ptolemy II. Figure 1 shows a simple example of a workflow described in GridNexus. The large frame on the right of the application window shows a workflow that calculates 3+4. The frame on the left-hand side of the window is a palette of modules that either perform operations or provide information. The modules can be dragged into the workspace on the right. A connection is made between two modules by clicking and dragging from an output port of one module to an input port of another. *Source* modules are modules that provide data but do not need an input connection. *Sinks* are modules that do not produce an output.

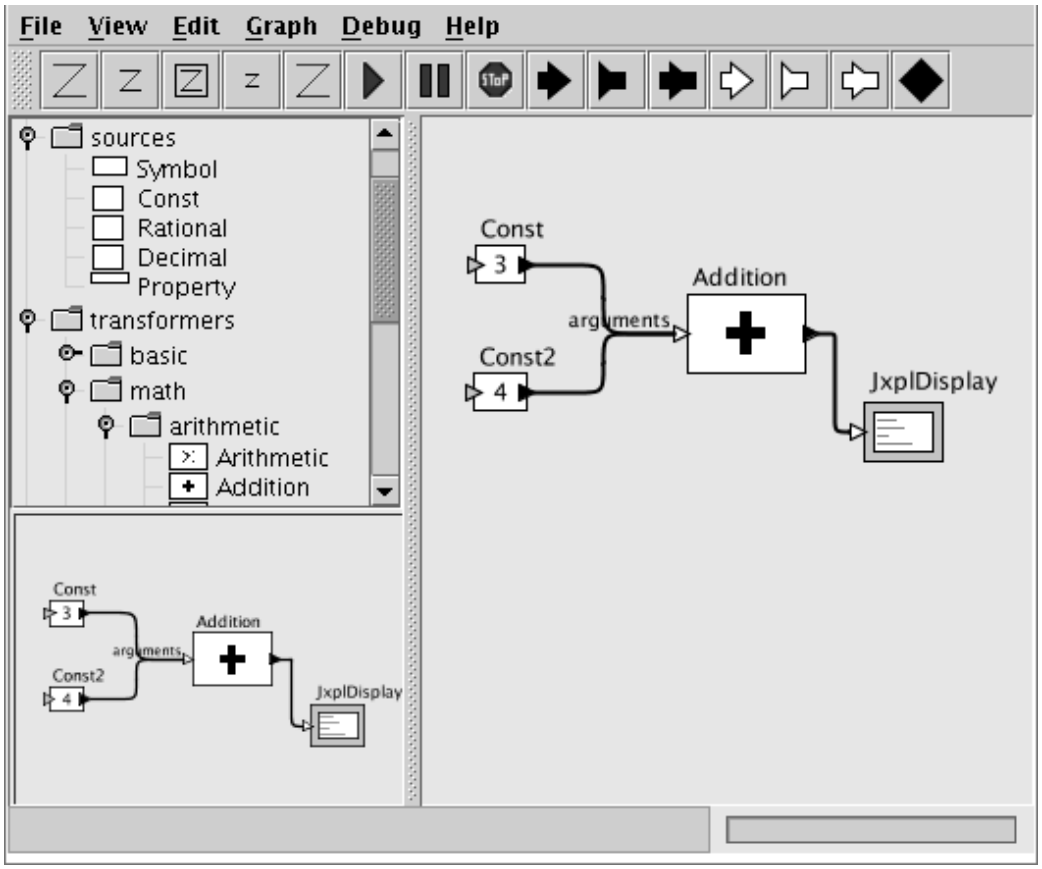

Figure 1: A simple workflow in GridNexus

GridNexus supports a wide variety of mathematical operations, and control nodes for branching and looping. A workflow can be used to define a function which can then be used in other workflows. There are also two features that allow the user to define functionality: module composition and function definition. Function definition is discussed in detail in the next section.

The module labeled "Addition" in Figure 1 is an example of a composite entity. It appears as one component in the workflow, but it actually represents a workflow that behaves like a user-defined function. In addition to allowing the user to build upon previously created workflows, the use of composites adds a layer of abstraction and leads to cleaner workflows. Composite entities can hide complexity from users who are not interested in that level of detail.

2.2 JXPL

JXPL is an XML-based scripting language inspired by LISP and written in Java [12]. There are several reasons why we chose to develop and use JXPL for this project:

1. The directed acyclic graphs created by the user in GridNexus are easily translated to the tree-like data structure used by JXPL. Thus, there is a natural correspondence between the workflows and JXPL scripts.
2. Since SOAP is the protocol used to transmit messages between grid services, XML is an obvious choice to represent a script or program to execute a workflow in a grid environment.
3. One of the most appealing features of LISP is the simplicity of its data structures. Everything is either an atom or a list. Since JXPL is based on LISP, both functions and data are represented in the same way: the all-inclusive list. This is a powerful feature of both LISP and JXPL that allows for implicit and dynamic allocation and de-allocation. It also allows for dynamic programming, recursion, and extensibility.
4. Other languages such as Perl work well with text, however the kinds of applications we are interested in (e.g., bioinformatics applications) often must deal with machine learning tasks and complex data manipulation activities (e.g., data mining). Like LISP, JXPL is well suited to these kinds of applications.

As an example of the ability of JXPL to handle machine learning, consider the problem of pattern matching. JXPL includes primitives that implement a unifying pattern matcher and deductive retriever. These are classic LISP applications (see the section on Logic Programming in [8]). For a simple example of what a deductive retriever does, consider a database that contains the following facts:

1. All dogs are mammals.
2. All beagles are dogs.
3. Rover is a beagle.

A deductive retriever uses recursive backward chaining to derive conclusions from the facts. For example, it could conclude that Rover is a mammal.

We demonstrate the basic structure of JXPL using the simple workflow shown in Figure 1. Table 1 shows the LISP program and the corresponding JXPL equivalent to compute 3+4.

In JXPL functions are represented by <primitive> tags. The JXPL processor receives an XML DOM element which can be any one of the simple types or a list. The DOM element is unmarshalled to a corresponding Java class – a subclass of JxplElement. JXPL includes the standard mathematical functions and uses the BigInteger and BigDecimal Java API classes to support arbitrary precision decimal calculations and exact rational arithmetic.

JXPL is designed for extensibility. There are two ways to add new primitives to the language: 1) by defining a new Java class to implement the primitive or 2) defining a new function using a primitive called "Defun." Defining a new primitive operation using Java is simply a matter of writing a class that implements the method "evaluate" with the signature:

JxplElement evaluate (JxplElement input).

Defining a new function using Defun is similar to defining a new function in LISP. For example, we can have a JXPL script which defines a function called "myadd," that takes x and y as parameters and returns the sum of x and y. Once the "myadd" function is defined, then another script can use it as if there was a primitive named "myadd" as shown in Table 2.

Table 1: A simple LISP example and the corresponding JXPL scripts

LISP	JXPL
(+ 3 4)	<list> <primitive name="Arithmetic"> <property name="operation" value="add"/> </primitive> <integer value="3"/> <integer value="4"/> </list>

Table 2: Defining new primitives in JXPL

Defining a function

```
<list>
    <primitive name="Defun"/>
    <symbol name="myadd"/>
    <list>
        <symbol name="x"/>
        <symbol name="y"/>
    </list>
    <list>
    <primitive name="Arithmetic">
        <property name="operation" value="add"/>
    </primitive>
    <symbol name="x"/>
    <symbol name="y"/>
    </list>
</list>
```

Using a function

```
<list>
    <primitive name="myadd"/>
    <integer value="3"/>
    <integer value="4"/>
</list>
```

The example above demonstrates the ability of JXPL to define simple functions, which allows for flexibility and extensibility. One of the drawbacks of LISP is its lack of readability. Because JXPL scripts are written in XML, they are even more difficult for people to read and write. However, JXPL is not intended to be a programming language for humans, but rather to be automatically generated and used by tools such as GridNexus.

2.3 Integrating GridNexus and JXPL with Grid Services[2]

The main goal of GridNexus is to create workflows that can make use of web and grid services. To accomplish this, we have implemented primitives in JXPL that are generic web and grid clients. These clients must inspect the stub classes or WSDL of the service to determine its interface.

First, we created a *GSClient* module in the GridNexus GUI, whereby the user can specify the factory URL, the instance name of the service, the stub class, and the port type. The GSClient module inspects the stub class to dynamically configure ports for each method of the grid service. Figure 2 shows the new module after inspecting the stub class. When data is passed in to one of the ports, the module creates the JXPL to call that service method with the data as arguments. The figure demonstrates a call to the "add" method of our counter service with an argument of 5.

The JXPL script includes a call to a *GSClient JXPL primitive* with properties that specify the factory URL, the instance name of the service, the stub class, and the port type. The service method and the arguments are given as arguments to the GSClient primitive. The primitive uses the OGSIServiceGridLocator to find the grid service and invoke the appropriate method with the arguments. The results, if there are any, are collected and returned by the GSClient primitive which is send to the output port of the GSClient module.

Integrating GridNexus with web services is even easier. The *WSClient module* only needs the URL of the WSDL. Through inspection of the WSDL, the WSClient is able to configure the ports and generate the JXPL to call a *WSClient primitive*. The primitive then generates a call to the appropriate method of the web service and returns the results.

2.4 Data Access and Integration

OGSA-DAI Grid Data Services are designed so that the output of one can be delivered to another. An individual GDS can be invoked through an XML script. However, the OGSA-DAI scripting environment is not rich enough to control interaction between services. For example, one cannot control the interaction of two Grid Data Services with only one OGSA-DAI script. Consider a generic example in which a source GDS will deliver its output to a sink GDS. Here is a typical sequence of steps that the client program takes:

1. Use a GDS factory to create the sink service instance and store the handle of the instance.
2. Run methods of the sink service in a separate thread so that it can wait for input.
3. Create a script to control the source service. This script needs to be created dynamically because it must contain the handle of the sink service.
4. Use a factory to create the source service and submit the control script.

A JXPL program can carry out the steps required to control the interaction of Grid Data Services. A variety of other languages could serve as well. In particular, the OGSA-DAI distribution has examples of Java programs for this purpose. However, GridNexus allows non-

[2] Grid services are like web services [3].

programmers to create JXPL to control GDS interaction in a graphical environment. The workflow described above as been implemented using GridNexus. Figure 3 shows an example of using an OGSA-DAI service.

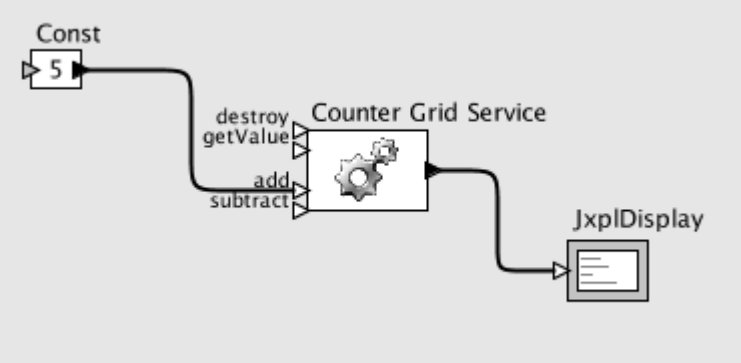

Figure 2: Using the new grid service client

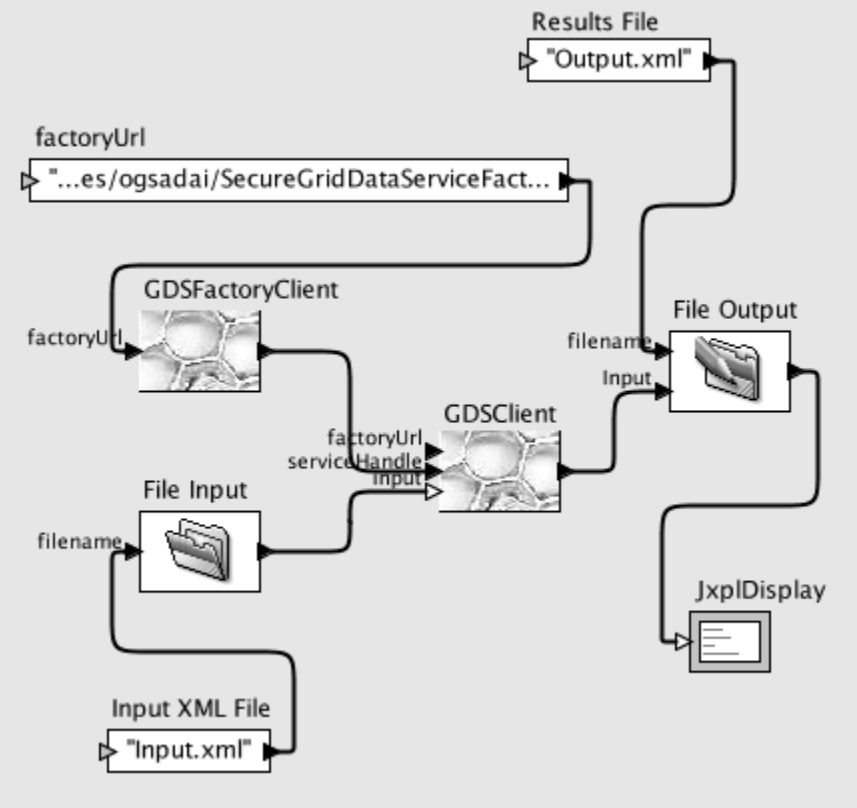

Figure 3: Using OGSA-DAI grid service clients

3. Proof-of-Concept

3.1 Background

We are working with faculty in biology, computational chemistry, mathematics, and e-business to make discipline-specific workflows that are useful for their research. This will enable them to create input files, submit jobs to nodes on a grid, and to view output results (completely transparently to the user) through a graphical user interface. An interdisciplinary faculty and student research team have collaborated to develop several GridNexus applications over the past year. Two interesting grid-enabled applications that we have designed and implemented are described below.

3.2 Biology Application

Biologists have generated large amounts of heterogeneous data in recent years [13]. Computational skills are uncommon among both biology students and faculty. Thus, using these large data sets requires substantial investment by a biologist in either additional training or salaries for programmers. This barrier means that exploratory data analysis is rare, and use of high-performance computing is limited to specialized groups who invest in their own IT management and clusters.

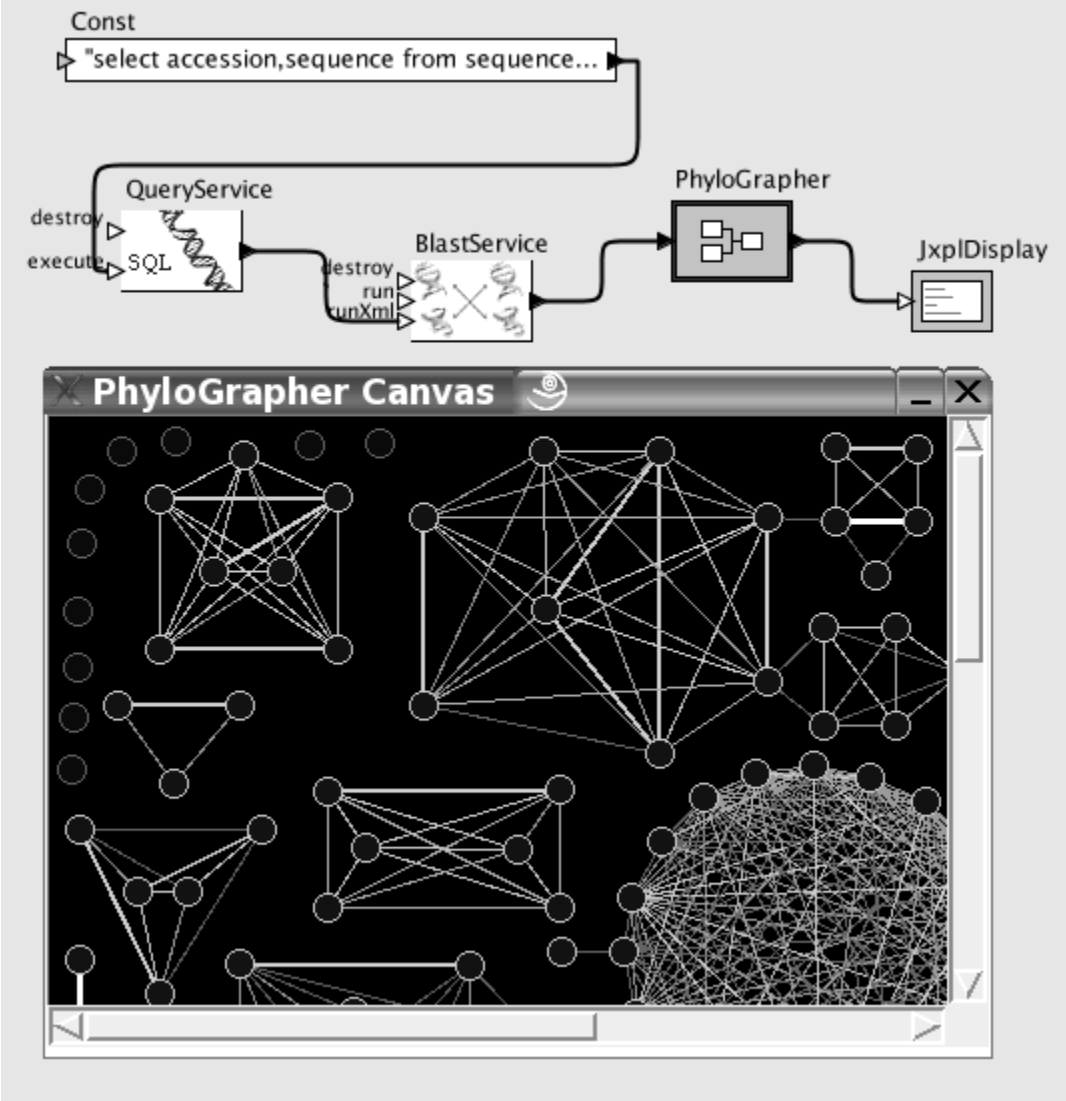

Figure 4: Molecular biology workflow in GridNexus

We have demonstrated with an earlier prototype [5] and with GridNexus that a high-level graphical interface allows access to sophisticated data mining algorithms by biologists without programming skills. In tests with students, the graphical interface is clearly less intimidating than command line instructions, and allows set-up of dataflows quickly and easily. Commercial software for data mining such as Enterprise Miner (SAS) and Clementine (SPSS) includes high-level graphical interfaces. There is no debate that graphical interfaces are needed, and in our development and testing of GridNexus we show that our flexible, layered interface is an excellent design for this problem. The ability to integrate existing software written in Java, C, or Fortran and the ability to have both interactive and batch processing are key advances over current open-source and commercial systems.

Figure 4 shows a GridNexus workflow for the DNA query and BLAST search example described in section 1. The boxes labeled "Database Query" and "Blast Service" represent OGSA Grid services deployed under Globus Toolkit 3 (GT3). These modules were produced using the generic GSClient described in section 2.3. The QueryService sends the query string to a grid service on one machine and the results of that query are sent to a BLAST service running on another machine. The BLAST service runs on a cluster of computers and performs a computationally intensive many-to-many search to produce a matrix of all possible comparisons. This matrix is then provided to a visualization program called PhyloGrapher [14]. This example shows the ease with which data flows between processes, even when those processes reside on separate machines. Compare this interface to that of a command line interface using FTP to transfer files between machines.

Searching for interesting patterns in data sets is the first step in developing experimentally testable hypotheses. The biologists in our group are very enthused about using GridNexus for computationally intensive data mining. It would not be possible for UNCW biologists to do this level of genomics data analysis without GridNexus.

3.3 Chemistry Application

One difficulty facing computational chemists is the incompatibility among file formats of molecule files used by chemistry software. The list of file formats includes ENT, DAT, MOL, among others. Most of these formats, although not all, use Cartesian coordinates. However, chemists often translate between file formats by hand in order to use a molecule created by one chemistry software program in another program. There currently are no tools to translate these file formats available to chemists. Furthermore, there is no consistency or standard used by software manufacturers for the same file type. One MOL file may differ in format from another.

We have added modules to the GridNexus library that translate between file formats as well as modules to perform some manipulations of the molecules. Figure 5 shows an example of a workflow that performs some operations on a molecule to produce a new one. The workflow reads in a molecule of one file format type, runs a molecule visualization program that allows the chemist to identify certain atoms of interest, sends this information to a program that performs some transformations on the molecule, and then the results are stored in a different file format.

The Jmol Viewer is an application that draws the molecule in a window and allows the user to rotate and zoom the molecule as well as to select parts of it. This demonstrates two useful features of GridNexus: 1) the ability to plug in off the shelf existing applications, and 2) the ability to incorporate interaction with the user in a workflow. We are currently in the process of creating a grid service to run the Gaussian program. Gaussian is one example of an application that is difficult and expensive to license. The high-level graphical interface of GridNexus will aid chemists in using this application and others.

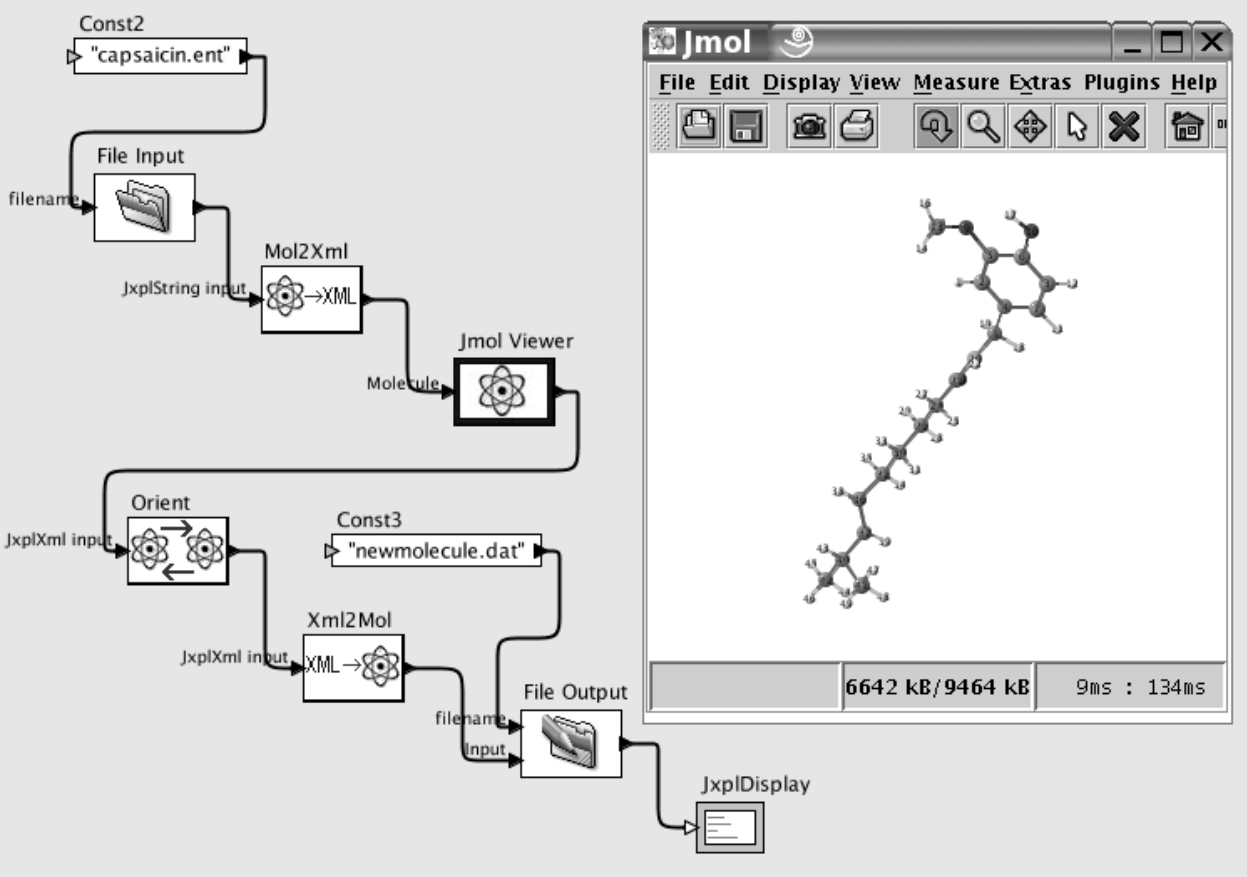

Figure 5: Molecular chemistry workflow in GridNexus

4. Future Work and Conclusions

We plan to use GridNexus to help create and deploy new grid services in addition to scripting existing services. We are also developing a generic module to provide interactive feedback while executing a workflow. It has become apparent from our discussions with users that some feedback is necessary between the user and the workflow.

Although there are several projects that are building tools for scientific workflows, GridNexus has the unique feature of separating the GUI from the execution of the workflow. New functionality can be added by creating new primitives in Java, or by defining functions in JXPL. The execution of the script may be carried out locally by the GUI or remotely as a grid service. Exposing the JXPL processor as a persistent grid service allows access to previously defined functions.

We are in the process of disseminating GridNexus to the wider research community. As part of this effort, we will be using GridNexus to facilitate the use of the North Carolina statewide grid project. We are working with faculty around the state to create workflows that use computer resources on the North Carolina Research and Education Network. GridNexus has also been used in a multi-campus grid computing class, where students were asked to create grid services and workflows that use them.

Finally, we are working on the Paraguin Project [15], whose goal is to build an open-source parallelizing compiler that can adapt a sequential program to a distributed system. The Paraguin compiler generates an MPI program. MPI is becoming the de facto open-source standard for *grid-enabling* applications for use in distributed environments. Our plan is to use the GUI to allow the user to import existing application code, send it to the Paraguin compiler to create a parallel version, and then to expose the parallel version as a new Grid service.

Acknowledgements

This work was supported in part by the University of North Carolina Office of the President and the Division of Information Technology Systems at UNC Wilmington.

References

[1] "Workflow in Grid Systems: Final Program", *Global Grid Forum*, Berlin, March 9, 2004. Retrieved from http://www.extreme.indiana.edu/groc/ggf10-ww/ on September 22, 2004.

[2] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration". Retrieved from http://www.globus.org/research/papers/ogsa.pdf on September 22, 2004.

[3] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, "The WS-Resource Framework", Version 1.0, March 5, 2004. Retrieved from http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf on September 22, 2004.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool", *J. Mol. Biol*, vol. 215, pp. 403-10, 1990.

[5] T. C. Hudson, A. E. Stapleton, and J. L. Brown, "Codifying Bioinformatics Processes without Programming", *BioSilico*, vol. 2, no. 4, pp. 164-169, July 2004.

[6] "Scientific Workflows Survey". Retrieved from http://www.extreme.indiana.edu/swf-survey/ on September 22, 2004.

[7] Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludaescher, and S. Mock, "Kepler: Towards a Grid-Enabled System for Scientific Workflows", *Global Grid Forum*, Berlin, March 9, 2004. Retrieved from http://www.extreme.indiana.edu/groc/ggf10-ww/kepler_towards_grid-enabled_system_for_scientific_workflows__bertram_ludaescher/kepler-GGF10.doc on September 22, 2004.

[8] "Ptolemy II". Retrieved from http://ptolemy.eecs.berkeley.edu/ptolemyII/ on September 22, 2004.

[9] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. C. Hong, B. Collins, J. Davies, N. Hardman, G. Hicken, A. Hume, M. Jackson, A. Krause, S. Laws, J. Magowan, J. Nowell, N. Paton, W., D. Person, T. Sugden, P. Watson, and M. Westhead, "OGSA-DAI: Two Years On," *Global Grid Forum*, Berlin, March 9, 2004. Retrieved from http://www.ogsadai.org.uk/docs/OtherDocs/15-Antonioletti-OGSA-DAI-DA-WS-final.pdf on September 22, 2004.

[10] "The Taverna Project". Retrieved from http://taverna.sourceforge.net/ on September 22, 2004.

[11] "Open Grid Computing Environments". Retreived from http://www.ogce.org on September 22, 2004.

[12] C. Hunt, C. Ferner, and J. Brown, "JXPL: An XML-based Scripting Language for Workflow Execution in a Grid Environment", *2005 IEEE Southeastern Conference*, Ft. Lauderdale, FL, April 8-10, 2005.

[13] Stein, L. D. "Integrating Biological Databases", *Nature Reviews Genetics*, vol. 4, pp. 337-345, 2003.

[14] "PhyloGrapher - Graph Visualization Tool". Retrieved from http://www.atgc.org/PhyloGrapher/PhyloGrapher_Welcome.html on September 22, 2004.

[15] C.S. Ferner, "The Paraguin compiler---Message-passing code generation using SUIF", in the *Proceedings of the IEEE SoutheastCon 2002*, Columbia, SC, April 5-7, 2002, pp. 1-6.

Jeffrey L. Brown is a professor in the Department of Mathematics and Statistics at the University of North Carolina Wilmington. His research interests include parallel and distributed computing, computer-aided geometric design, and computational geometry. He has developed and teaches an advanced Web programming course that includes topics such as JSP/JDBC, XML, XSLT, Web and Grid services.

Clayton S. Ferner is an assistant professor in the Department of Computer Science at the University of North Carolina Wilmington. His research interests are parallel and distributed computing, grid computing, and compilers for parallel and distributed computing. Ferner is a member of the IEEE Computer Society and the ACM.

Thomas C. Hudson an assistant professor in the Department of Computer Science at the University of North Carolina Wilmington. His research interests span computer graphics, scientific visualization, distributed systems, networking, data mining, and collaborative computing. He designs new tools for scientists and engineers using techniques from distributed systems & computer graphics.

Ann E. Stapleton is an assistant professor in the Department of Biology and Marine Biology at the University of North Carolina Wilmington. Her research interests include genetic and genomic analyses of plant ultraviolet radiation responses, and plant functional genomics/bioinformatics, along with advancement of undergraduate research participation.

Ronald J. Vetter is a professor and chair of the Department of Computer Science at the University of North Carolina Wilmington. His research interests include parallel and distributed computing, mobile computing, and wireless networks. Vetter is a member of the IEEE Computer Society and the ACM.

Tristan M. Carland is a 2005 graduate of the University of North Carolina at Wilmington with a B.S. in Computer Science and Marine Biology who will be attending the Scripps Institute of Oceanography in the fall of 2005. His interests include data mining, gene regulation networks, distributed computing, Nemertean diversity and sailing.

Andrew Martin is an undergraduate student majoring in computer science at the University of North Carolina Wilmington. His main research focus is on computational chemistry and its integration with GridNexus.

Jerry Martin is an undergraduate student majoring in computer science at the University of North Carolina Wilmington. His main research focus is on computational chemistry and its integration with GridNexus.

Allen W. Rawls is an undergraduate student at the University of North Carolina Wilmington pursuing a degree in Computer Science with a minor in Physics. In addition to grid computing, Mr. Rawls's interests include database systems, data security and aeronautics with a passion for flying.

William J. Shipman is an undergraduate student majoring in computer science at the University of North Carolina Wilmington. His research interests include parallel computing, grid computing, bioinformatics, web-based application development and deployment.

Michael Wood graduated from the University of North Carolina Wilmington in May 2004 with a B.S. in Computer Science. He is currently a graduate student at North Carolina State University working to create biological grid applications.