2007

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

Extending GridNexus and JXPL to Support the Visual Assembly of Web Services

By

Eric C. Harris

University of North Carolina Wilmington

Wilmington, NC

July 2007

Capstone Project Committee:

Dr. Ron Vetter (chair)
Dr. Jeff Brown
Dr. Thomas Janicki

Capstone project submitted as a portion of the requirements for the Master of Science Degree in Computer Science Information Systems

Table of Contents

Abstract

This paper describes a set of extensions (primitives, bindings and graphical modules) and their implementation to facilitate the visual assembly of web services from scientific workflows within the GridNexus computing environment. The implementation approach for this work involved employing a strategy for defining WSDL message types and procedures such that each will be bound to a defined workflow for evaluation. Because of its Axis2 foundation, web services produced by GridNexus can run under any J2EE application container by providing the service as a deployable archive file. The paper concludes with a discussion of various design alternatives including the advantages and disadvantages of each.

1. Background

1.1 What is a Service Oriented Architecture?

A Service Oriented Architecture (SOA) is a methodology of defining a computational process as the interaction of smaller reusable services within a network. Interactions may involve simple data passing with a single service or complex interactions among multiple services to coordinate some activity [1].

Hewlett Packard describes five key benefits to adopting a SOA. These benefits are described as visibility, manageability, reusability, adaptability and interoperability. Making components visible and manageable revolves around having single control points of information that are adopted to reduce redundant and conflicting information. Reusability and adaptability are a result of having resources available in such a way that they can be used in a variety of different processes [2]. Interoperability is provided by using commonly accepted standards and protocols.

The ideas behind SOA are not new. Distributing processes across a network have been attempted with proprietary remote procedure call (RPC) technologies such as Common Object Request Broker Architecture (CORBA) and Microsoft's Distributed Component Object Model (DCOM). These systems were often platform dependent and enabled limited interoperability. Interoperability, in the context of this paper, describes the ability to communicate not only between different hardware configurations and operating systems, but also to interoperate between communications packages written by different vendors. This is an area where CORBA and DCOM fell short. With SOA, interoperability is provided by open standards and protocols such as the Hypertext Transfer Protocol (HTTP) and extensible Markup Language (XML).

1.2 XML

XML is a simple, very flexible text format that is quickly gaining acceptance as a means of communicating a variety of data [3]. Because of the inherent flexibility of XML, most popular programming languages include API's for reading and parsing XML documents. This has driven XML to become a medium for sharing complex content between various applications and frameworks.

Because of the ability to enable interoperability, many XML based standards have been established for a wide variety of applications and domains. Two prominent standards bodies for XML are the Organization for the Advancement of Structured Information Standards (OASIS) and the World Wide Web Consortium (W3C). A few of the standards are listed in the table below.

Title	Standards Body	Year	Domain
Web service-Business Process Execution Language (WS-BPEL)	OASIS	2007	Web/SOA
Open Document Format v 1.1	OASIS	2007	Office
Electoral Markup Language (EML) v 4.0	OASIS	2006	Government
eBusiness eXtensible Markup Language (ebXML) v 2.04	OASIS	2006	Business Transactions
Simple Object Access Protocol (SOAP) v 1.2	W3C	2006	Web/SOA
Web service Description Language (WSDL) v 1.1	W3C	2001	Web/SOA
Scalable Vector Graphics (SVG) v 1.1	W3C	2003	Graphics

Table 1. Sampling of XML Standards

XML formats are often described by a schema using either a Document Type Definition (DTD) or XML Schema Definition (XSD) document(s). DTD describes the XML in a flat file document whereas the newer XSD format describes the XML schema in its own XML format. Failure of the XML to meet this schema can result in failure of the document to be read and interpreted properly. According to the introduction on the XSD schema specification, XML schemas express shared vocabularies and allow machines to carry out rules made by people [4]. A schema includes constructs to describe custom data type descriptions, formatting, optional/required attributes and elements, and minimum/maximum boundaries on the number of elements that are permitted [4].

1.3 Web Services

Many of the SOA technologies today rely almost exclusively on web services. Web services are network applications that rely on standard protocols for exchanging information between applications and organizations [5]. This exchange is driven by the use of the XML to encode data objects and send them using HTTP. Web services promise interoperability in heterogeneous computing environments by using open standards and loose coupling.

There are two methods in which web services may communicate. A web service may use either the formal Simple Object Access Protocol (SOAP) or the informal Representational State Transfer (REST) method.

1.3.1 Representational State Transfer

REST services send XML via the GET, PUT and POST requests found within the HTTP protocol [6]. Roy Fielding, who introduced REST in his Ph.D. thesis, described it as operating like a well-defined web application. Similar to a web page and browser, a REST service provides the client with a representation of an object. Information about the object is maintained in the state of the client. As the client follows links, it transfers the state of the representation for a new one [6]. There is no standard for REST, as it is more a style of communicating with XML over HTTP than a formal definition [6]. REST is popular and has gained a foothold thanks to its small learning curve and simplicity. Some have labeled it as being the “grassroots” approach to web services [6]. Advocates of REST tend to consider SOAP as being overly complex and bloated [7]. With this in mind, Amazon has stated that the majority of their web services users prefer to use REST instead of SOAP [26]. Providers such as Amazon and E-Bay offer both

REST and SOAP API's for their users. However, an analyst at ZapThink recently stated that although REST services work well in isolated instances, they miss the bigger picture of web services where interoperability with other vendors is the goal [7]. REST works great in simpler instances but more complex ones involving service oriented architectures tend to use SOAP [7].

1.3.2 Simple Object Access Protocol

The SOAP specification is maintained by the World Wide Web Consortium (W3C). SOAP defines the use of XML over HTTP to exchange information with services in a language/platform-independent manner. SOAP provides a modular method for describing application semantics (http://www.xml.org/xml/resources_focus_soap.shtml). SOAP contains an envelope with a header and a body. The header maintains information regarding the authentication mechanism, optional entries and/or other pertinent information to the application [9]. The body of the SOAP message is the encoded data being sent. The format of the body is abstract and defined for the respective service using the Web service Definition Language (WSDL).

The WSDL, also maintained by the W3C, is an XML document that describes the interface that the service must implement and clients invoke. A web service's WSDL defines data types, operations, messages of the operations and SOAP bindings to be used when communicating across the network [10]. Many WSDL documents are now available. Parties needing to discover how to invoke a particular service can use the Universal Description, Discovery and Integration (UDDI) standard for discovering services [11].

1.4 Multi-Programming Language Support for Web Services

Web services are network applications that rely on standard protocols for exchanging information between applications. Web services use the open standards of XML and HTTP to interoperate between organizations [10]. Many APIs have been produced to extend web service support into a variety of different languages. For example, The Apache Foundation's Axis Project is an open source API extending web service support to Java and C++. Meanwhile, Microsoft has been increasing including web service support in the .NET API for C#, VB.NET, and like the Apache Foundation, C++.

1.5 Apache Axis2

One open source project for supporting web services in Java is Axis2, from the non-profit Apache Foundation. Axis2 was contributed to by volunteers from large technology organizations such as IBM. Axis is currently used by many J2EE vendors to provide web service support in their application servers. Examples include Apache Tomcat J2EE, IBM Websphere, and Apache Geronimo.

The structure of Axis2 is modular and capable of handling multiple formats and patterns of communication. This modularity allows Axis2 to parse both SOAP and REST messages. Message Exchange Patterns (MEPs) enable the service to communicate either synchronously or asynchronously between service and client.

The Axis2 API consists of three internal models. The first is the XML Processing Model, which is responsible for parsing and binding of the messages into objects representation for later processing. Once beyond the XML binding, the SOAP

Processing Model then handles decoding and encoding the layers of the SOAP message. Axis2 offers the ability to layer SOAP messages through a chain of handlers. Each "handler" can include capabilities such as the WS-Security framework or any other custom written extension.

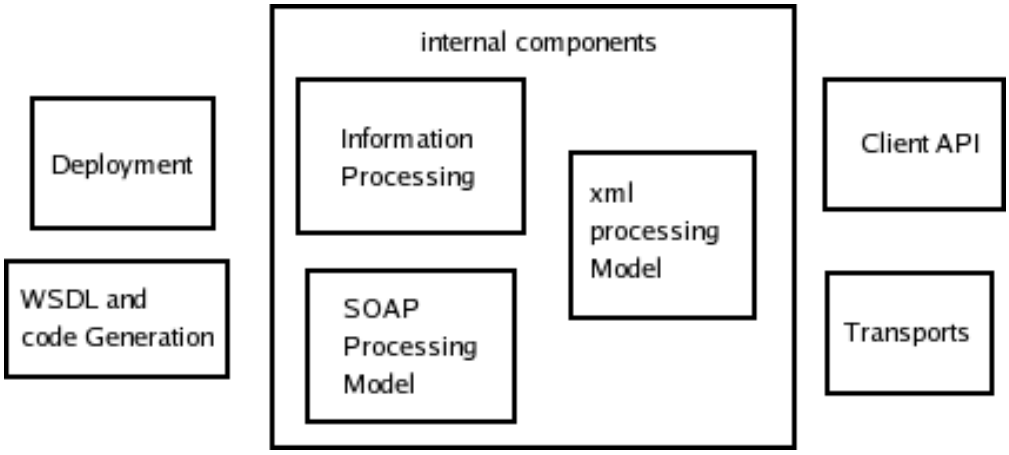


Figure 1. Core Components of Apache Axis2
(Courtesy Apache Foundation http://ws.apache.org/axis2)

Ultimately the messages will find their way to a message "receiver" which will take the contents of the message and perform the "logic" required to fulfill the invocation request. The output is then forwarded though a similar set of handlers that wrap it into a response message with the appropriate XML envelopes to send back to the client. The state of the system is contained in the messages being sent to and from the service. As messages are passed through the chain of handlers the information is held by the Information Processing Model.

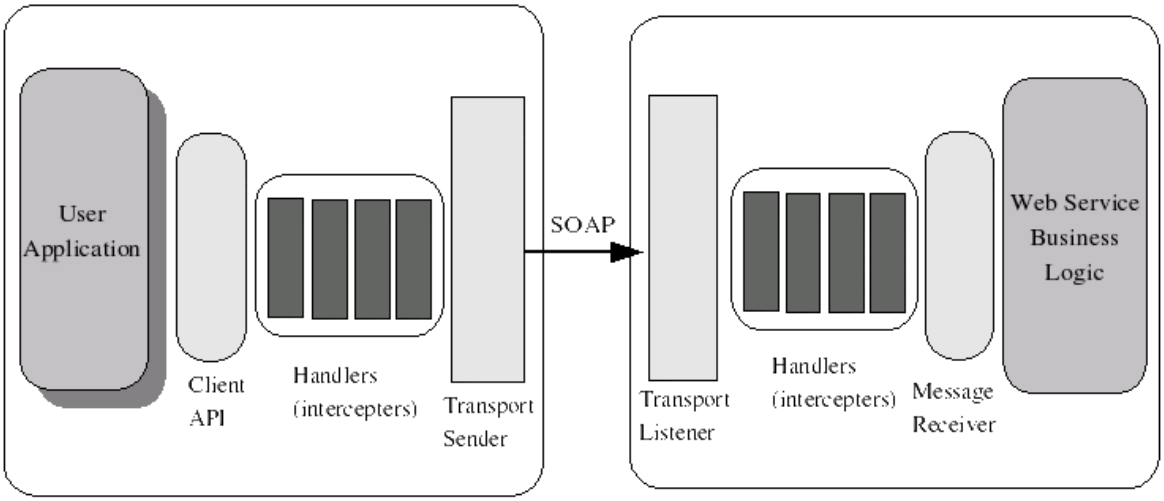


Figure 2. SOAP Processing through a Chain of Handlers (Courtesy Apache Foundation http://ws.apache.org/axis2)

The Information Processing Model spans five layers in Axis2. These layers are: ***Axis2 Configuration -> Service Group -> Service -> Operation -> Message*** Each level contains two sets of information. The first is for storing relevant information in contexts while the other is for system configuration/description.

- Configuration Layer – This layer of the Information Processing model maintains runtime stats and global configurations applicable to the Axis2 engine. Examples of configurable parameters involve transport mechanisms (Standalone HTTP Engine, Java Mail System, J2EE Servlet, etc), and services to be loaded.
- Service Group Layer – While the Axis2 Configuration Layer maintains information pertaining to the global system, the service group layer narrows its focus on specific groupings of services. Examples of parameters at this level tend to focus on deployment information for the service group.
- Service Layer – Narrowing the focus even more, an internal service layer enables

description of the service including its constituent operations.

- Operation Layer – The Operation Layer provides the information necessary to describe an operation. This description details the chain of handlers and receiver used during the invocation of the operation. The receiver selected must implement how to invoke the "logic" of the operation and describes the message pattern in which the operation uses. An example of a built in receiver is org.apache.axis2.rpc.RPCInOutMessageReceiver, which performs the basic "invoke and wait for response" style of function call.
- Message Layer – The final layer in the model focuses on the message. There is no configuration to specify here. This layer simply stores the state of the object in a context for use in the receiver.

1.6 Business Process Execution Language

The Business Process Execution Language or BPEL is a published standard for connecting SOAP web services into a single orchestrated work flow. BPEL is an imperative style XML language that is highly formalized and specifies constructs for loops, conditionals, fault tolerance, parallel execution and variables. Each web service is bound onto a “partner link” which allows the engine to maintain connectivity and role of information between various web services [12]. Once a BPEL script is specified, it is then deployed and bound as a web service itself. Clients needing to use that process communicate only with the new BPEL service instead of invoking each individual service required by the process. From a maintainability standpoint, BPEL creates a single point in which modifications to a process may be applied for all consumers. BPEL version 1.1 (more formally BPEL4WS) was published as an open standard in May of

2003 and has since been implemented by several vendors such as IBM, JBoss, Oracle and Active-Endpoints [13]. On April 12, 2007, WS-BPEL 2.0 was voted as a standard by the OASIS international standards consortium. WS-BPEL 2.0 consists of many significant changes over the previous BPEL4WS 1.1 standard.

1.6.1 Features of BPEL

BPEL contains many constructs used to help describe workflows. They include mechanisms for fault handling, parallel execution, sequential execution, loops and conditionals. BPEL is an imperative language, meaning that all invocations require an assignment operation to some variable before being forwarded onto the next step in the workflow invocation.

The typical pattern of execution for a BPEL workflow is that of receive (input from web service call), invoke (logic defined by the BPEL process), and reply (back to the client). A WSDL file for the BPEL process is used to describe the “receive” and “reply” operations for external consumers. Internally these requests are parsed and forwarded to the internal logic for further processing.

For defining interactions during the “invoke” operation the following BPEL constructs are used [19]:

BPEL Tag	Description
Assign	Assign a value to a variable from an xml data type.
Sequence	Items to occur in a specified order
Flow	Perform items in parallel
If, ElseIf, Else	Perform a set of conditional operations
Pick	Event processing • OnAlarm – event triggered by a timer • OnMessage – event triggered on inbound message
While/RepeatUntil	Manage a Loop
Throw/ReThrow	Throw a fault on an error condition
Exit	Exit the process
Wait	Delay execution

Table 2. BPEL Tags and Descriptions

1.6.2 Industry use of BPEL

BPEL has been adopted by many organizations for the description of workflows between components in service oriented architectures. Many organizations today use BPEL to map business processes to solve a variety of tasks from managing transactions to creating new processes from legacy applications. Today products such as Microsoft BizTalk, IBM Web Sphere and many others are offering BPEL support as a part of their products. BPEL has become an important component in many SOA architectures by playing a vital role in fulfilling the SOA paradigm by offering an abstract way of defining how service components interact [20].

1.7 Overview of GridNexus and JXPL

1.7.1 GridNexus

In fields such as Computational Chemistry, researchers often have to use a variety of different and dispersed *nix based applications for scientific computations such as the "Gaussian" application. Because there exists many different input formats, file conversion programs must be run to ensure the proper conversion of the information in a format that can be parsed by the software application. Indeed, to solve many scientific problems, software applications are often combined and executed in sequence passing the output of one as the input of another. For the user, properly running these "workflows" is often cumbersome and prone to error. Today, many scientific computations are moving in the direction of *Grid Computing* which can be defined as "a network infrastructure for distributed computing".

GridNexus was created at the University of North Carolina Wilmington as an extension of Berkeley's Ptolemy project [16]. GridNexus supports the creation of scientific workflows in an easy to use graphical environment. GridNexus enables users to orchestrate grid services, web services, and more using simple configurable drag and drop boxes [17]. Using these boxes, users can develop and run complex processes, called workflows, without having to concern themselves with the syntactical details of code implementation. The GridNexus GUI acts as a graphical front end that creates script in an underlying XML based scripting language known as JXPL which is responsible for the workflow execution [18].

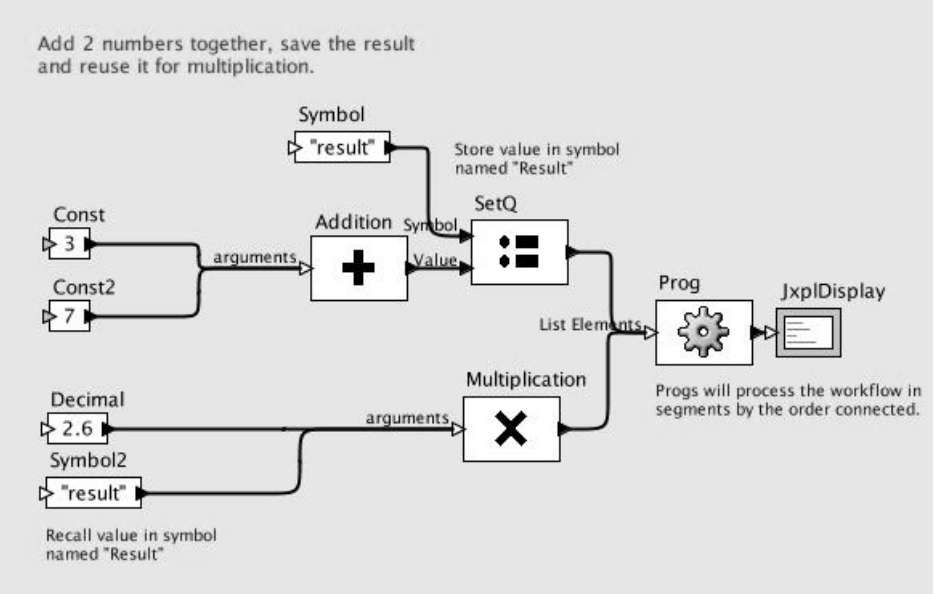


Figure 3. Sample GridNexus Workflow

Although GridNexus can consume web and grid services, it cannot currently define them. In instances where remote JXPL execution is required, GridNexus generates a JXPL script that is sent to a remotely bound JXPL engine. These engines are pre-bound using RMI, HTTP, or the JXPL grid service. This solution makes distributing a JXPL process easy, but is less than ideal when making the workflow available beyond GridNexus itself. In this paradigm, the client must be capable of dictating the script that the remote engine is to execute. Clearly, providing this level of strict control over the remote engine may not always be in the best interest of the organizations security practices as it may be possible to execute any arbitrary script or command on the remote system if not configured properly. Being able to visually define and create a web service from within the GridNexus graphical editor would greatly facilitate the development of scientific workflows. That is the aim of this project.

1.7.2 JXPL

JXPL is a Lisp inspired XML scripting language [15]. Like Lisp, the JXPL processor recursively evaluates lists that may contain other lists or atoms. Much of the functionality in JXPL comes from atoms called Primitives. Primitives may exist at the start of a list and define functions in the language. Examples of primitives would include the functions of "SetQ" or "Cons." Primitives either represent defined macros or are bound to pre-written Java code. JXPL is created by the GridNexus GUI and the JXPL engine responsible for its execution.

1.7.2.1 JXPL Bindings

JXPL includes a set of bindings that represent data structures in JXPL. The bindings are responsible for the linking the association between the XML code and the representation needed for computation. The Lisp the counterpart to a binding would be an "atom" [14].

Binding	Description
String	Defines a String
List	Provides the generic Lisp structure for a List
Integer	Whole number
Rational	Representation of complex numbers
Decimal	Allows for non-whole numbers such as "2.5"
Symbol	Abstract place holder that can be substituted for during evaluation
Primitive	Placed at front of list and maintains a link to an executable Primitive (see below)

Table 3. JXPL Bindings

1.7.2.2 JXPL Primitives

JXPL Primitives are the executable instructions in a JXPL script. These actors can be seen as being analogous to a function as found in many imperative languages [14].

Primitives go at the front of a list and may accept following elements as parameters. The Primitive binding consists of a required attribute called “name.” Naming the primitive associates it with a pre-compiled byte code that the JXPL engine can invoke. The JXPL package contains a library of many pre-compiled primitives, some of which are listed here.

Prim Name	Description
Cons	(borrowed from Lisp) create a new list from a set of elements
Car	Take the first element of the list
Cond	Define a condition in which elements are to be compared
Prog	Break a list down into segments for evaluation. Also provides forking and the ability to forward to remote JXPL Engine.
SetQ	Specify what to substitute in place of a given symbol
.io.FileInput	Read data from a file
.io.FileOutput	Write to a file
.ws.WSClient	Generic Client for invoking a Web service
.ws.WSRFClient	Generic Client for invoking a Grid Service (using WSRF specification)
Fault	Throw a fault during evaluation
HandleCase	Catch a fault
LocalExec	Invoke an application
Arithmetic	Basic mathematical functions

Table 4. JXPL Primitives

Each primitive may also allow for a set of properties to be associated to it. For example the Arithmetic primitive may accept a property called “operation” which may consist of value such as “add”. An example of a complete JXPL Script is as follows.

```
<list>
        <primitive name=”Arithmetic”>
                <property name=”operation” value=”add”/>
        </primitive>
        <integer value=”2”/>
        <integer value=”5”/>
</list>
```

1.7.3 Runtime Architecture

After the user creates a workflow and hits the execute button. The GUI will then generate a JXPL Script represented as an abstract JXPL element (Java object). This element will then be forwarded to the JXPL processor which be recursively evaluated from the bottom up.

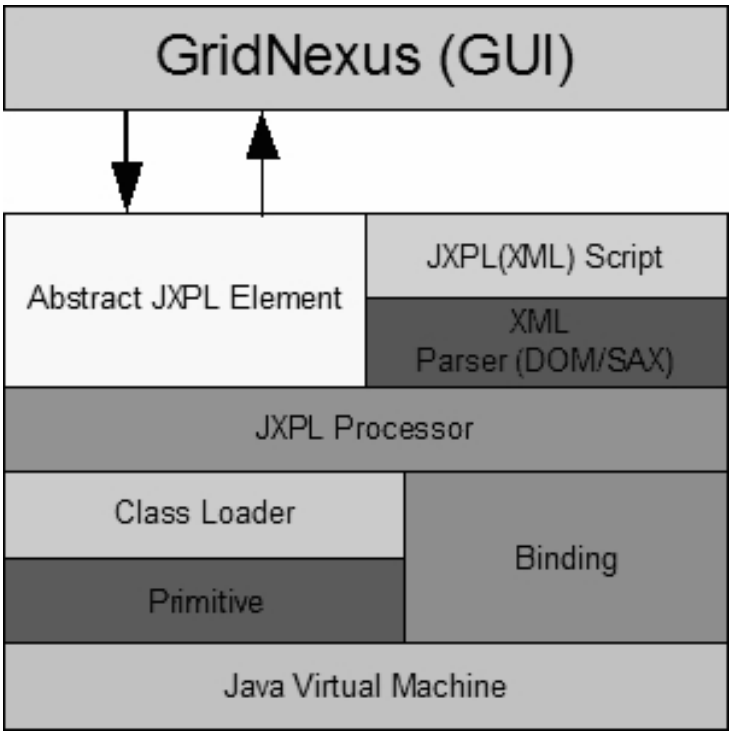


Figure 4. JXPL Architecture

The JXPL Runtime architecture accepts scripts as either XML or an enveloping abstract Java object. The function of the "JXPL Processor" is to associate the scripts with their corresponding byte code representation for the Java virtual machine. In the event of primitives the processor will examine the name of the primitive, and utilize the Java Class Loader to load the appropriate class. The Processor always assumes that the primitive is always going to be of the "org.jxpl.primitives" package. For example, the "Arithmetic" primitive is described by the "org.jxpl.primitives.Arithmetic" class in the Java class path.

1.7.4 Open Source

GridNexus and JXPL have been approved to be made available as open source software, denoting the move from a prototype to a production release [27]. By moving to open source, GridNexus will be made available to the community at large. Moving to open source will also help to further engage the user community [27].

The project was approved under a BSD license, similar to the license used with Ptolemy II. The BSD license provides the software "AS IS", and releases the organization producing the code from any liabilities associated with its use. Users of BSD licensed software are free to use the software in a variety of open source or proprietary projects, provided that the provided copyright notice stays at the top of every source file of the program or module[1].

[1] BSD License Information: http://www.opensource.org/licenses/bsd-license.html

1.7.5 Current User Groups

At the time of this writing, there are currently three groups in North Carolina using and/or extending on GridNexus to in a variety of domains. The first group consists of various campuses in the UNC System using GridNexus as a tool for teaching Grid Computing. This is a continuation of a project run by Dr. Barry Wilkinson of the University of North Carolina - Charlotte and Dr. Clayton Ferner of the University of North Carolina - Wilmington. In the past, the class has enrolled as many as 40 students, all of which were required to use GridNexus for homework assignments [28]. The course has been funded by the National Science Foundation and has been broadcasted to as many as 12 campuses throughout North Carolina.

The second group of users consists of researchers within the UNC system where the tools being used do not always match up with the researcher's domain of expertise [15]. Many of the tutorials and examples associated with the GridNexus toolkit are from the fields of Computational Chemistry, and Bioinformatics. Presently, a team at North Carolina State University is working on using GridNexus to represent a series of workflows for conducting research in Evolutionary Biology. Ultimately they hope to make tools such as Mr. Bayes and other computational tools available on a cluster running grid middleware and use GridNexus to connect them together into coherent processes.

The third group using GridNexus consists of a private firm with an SBIR phase II government contract, exploring GridNexus and JXPL as a means of defining problems in a biologically inspired artificial intelligence infrastructure (e.g. membrane computing).

2. Open Source Visual Web Service Workflows

2.1 Criteria

Two toolkits met the following criteria for comparison to the solution presented in this paper. First, the toolkit must be capable of not only consuming web services but publish them in web service format consumable by another client. Second, the software must be open source (no stipulation on which open source license was specified).

From these criteria two software solutions surfaced: the OMII BPEL plugin for Eclipse and JOpera.

2.2 JOpera

JOpera is a similar workflow project developed by a team at the Swiss Institute of Technology in Zurich [21]. Like GridNexus, JOpera is a tool for rapid development of workflow processes incorporating a runtime environment and a graphical editor (JOpera).

JOpera is designed as a plugin for Eclipse and relies on many of the existing frameworks such as the GMF (Graphical Modeling Framework) plugin and many custom written modules. At runtime JOpera compiles Java byte code incrementally during execution [21].

For creating web services, JOpera provides a radio button at creation of the workflow that determines if it is to be published as a web service or not. If selected, then JOpera will automatically load the web service in its own custom WS Container run by default on port 8080.

2.2.1 Components

JOpera includes many components that may be used to add functionality to a process. These components include support for SSH, Unix Commands, Asynchronous Messaging, Web Services, Grid Services, and Java Snippets.

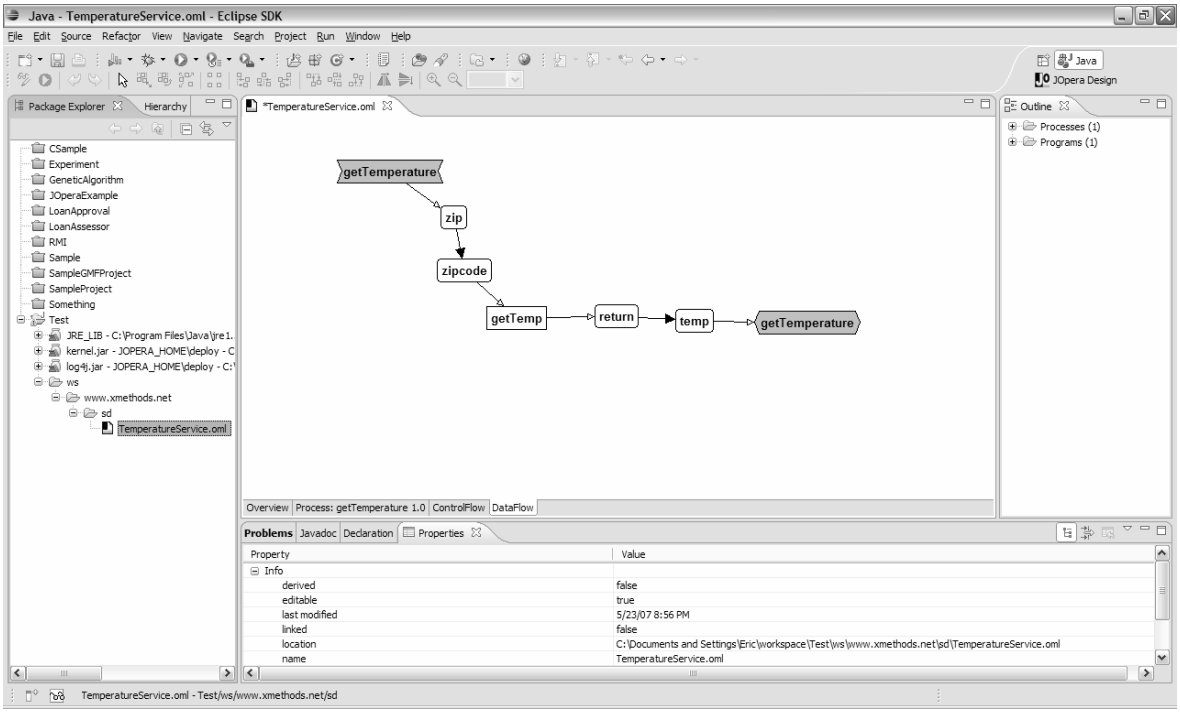


Figure 5. JOpera in Dataflow View, Displaying Flow for Invoking "Get Temperature" Service

2.2.2 Flow Specifications

JOpera consist of two types of flows that can be used together to create a workflow. These are Data Flow and Control Flow. These are both available as tabbed panes in the graphical editor.

The data flow tab provides a graph representation responsible for specifying the path in which data is to be forwarded through components in the workflow. The control flow tab on the other hand provides a directed graph structure controlling the direction in which to iterate through the components in the workflow.

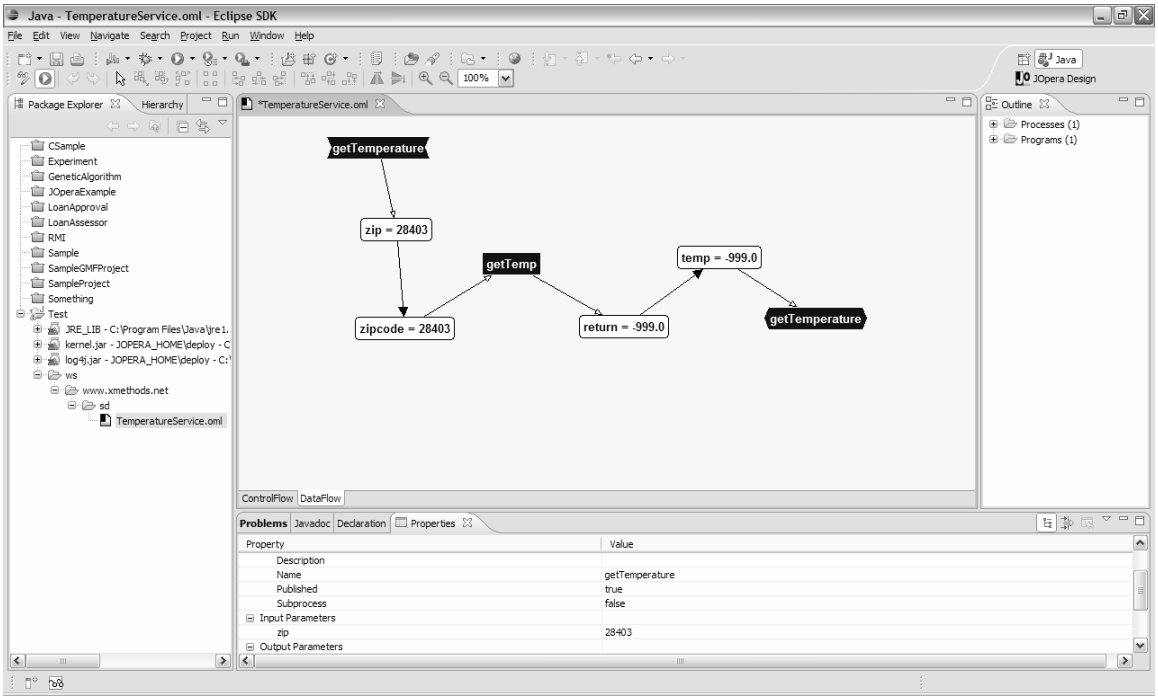


Figure 6. JOpera in Monitor Mode, Note the Display of Intermittent Values

2.2.3 Monitoring

Once a graph has been assembled, the workflow can be switched to monitoring mode at runtime. In monitor mode the various graph nodes will be highlighted as they are traversed by the execution engine.

2.2.4 Problems and Weaknesses

JOpera may be robust and flexible but it is less than ideal for a non-programmer end user. In order to include a web service for example, the user must go through a myriad of menus to create and configure the web service as a program. First the user has to go and import a WSDL file found on the Internet. Next the user must return into JOpera and create a new process in the process menu to create a new process from which the service will actually be defined. The user must then select "edit" process to arrive at a screen displaying options for naming and defining parameters for the "process." After this is complete, the user will be taken to a "data flow" graph containing components corresponding to the process and any programs that may have been defined. The purpose of which is to specify the path in which data will travel throughout the system. The user must expand each input/output property from each of the components and connect them accordingly. If the user wishes to establish conditional control this must be accomplished in yet another graph called the "control flow."

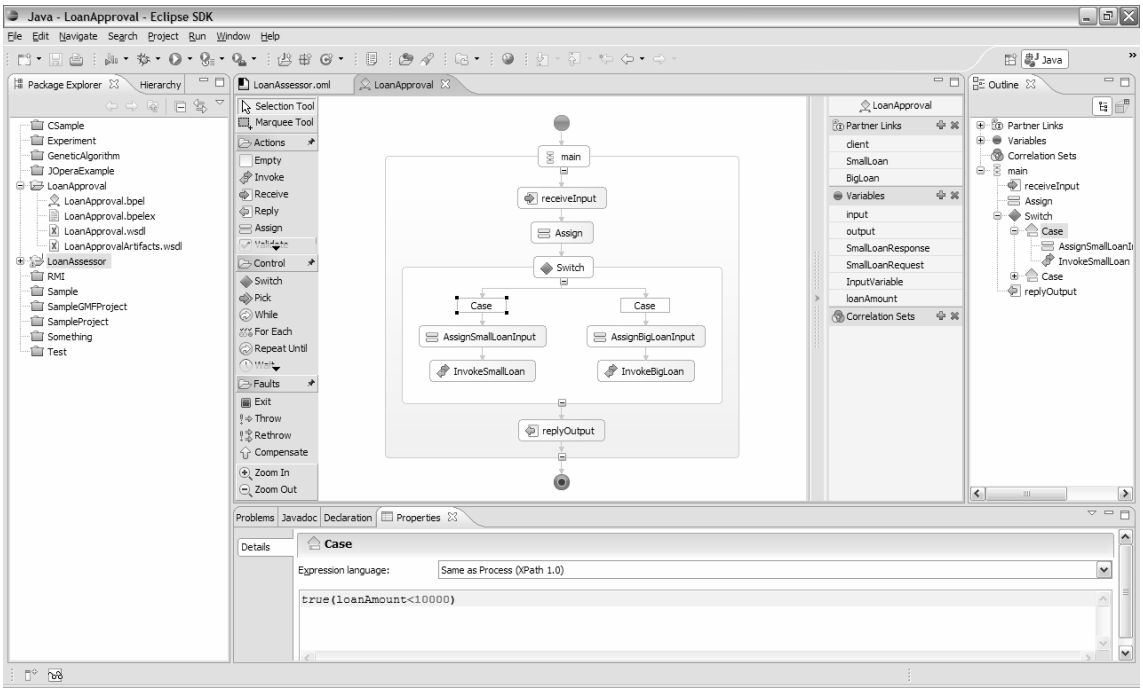


Figure 7. Eclipse BPEL Editor

2.3 OMII BPEL

Open Middleware Infrastructure Institute (OMII) BPEL is built upon two open source projects that have been developed for hosting BPEL processes. It utilizes both the Eclipse BPEL Project, and the open source ActiveBPEL engine. Seeing that these tools were being used to target business processes, the OMII group decided to extend and tie these two together in an attempt to use BPEL.

2.3.1 Eclipse BPEL Project

The BPEL Project is an Eclipse plug-in for developing scripts based on the latest BPEL 2.0 specification. It is being developed in collaboration as an open source editor

by IBM and Oracle. It features many drag and drop boxes that represent various tags found in the BPEL language. The tool also includes tools to help build XPath expressions for extracting the information required from multipart messages and assigning to other parts of the process. The editor is licensed under the Eclipse Foundation License.

2.3.2 ActiveBPEL

There are relatively few Open Source BPEL 2.0 engines available. One of the most prominent open source engines to date is the ActiveBPEL engine. This engine has been available since 2004 and based on the Active Endpoints website is purported to now support both BPEL 1.1 and the newly ratified WS-BPEL 2.0 specification as of version 3 [25].

2.3.3 Problems and Weaknesses

The BPEL Editor and ActiveBPEL together have key weaknesses. First, the BPEL editor is a plugin designed for Eclipse 3.2 or greater. Although this is a great strength it also is a weakness. For new users and non-programmers, using Eclipse involves a bit of a learning curve getting used to “perspectives” and figuring out which features they need and which to ignore. Aside from the learning curve associated with Eclipse, the BPEL editor is designed to be abstract and generate BPEL in general and as of June 2007 does not provide any explicit ties with any BPEL Engines. This is good in that it is loosely coupled and universal. However, this is bad when you want to make it work with a specific engine such as ActiveBPEL. In order to make it work with

ActiveBPEL, the user must provide additional XML configuration files to run the service. The OMII BPEL solution has made some strides at overcoming this deficiency by providing better linkage between the two. Also, BPEL can only consume web services. This may be a limiting factor for researchers who frequently need to incorporate specialized applications within their service.

3. Publishing Workflows as Web Services

An identified need in the user community provided the motivation for this project. GridNexus users expressed a need to visually assemble web services as this would greatly facilitate the creation and sharing of their scientific workflows. An evaluation of the previously described Web Service toolkits highlighted the need to provide researchers with the ability to create services themselves. It was determined that GridNexus was an ideal and extensible platform to provide this functionality.

3.1 Benefits

GridNexus has the capability of producing workflows that can do a variety of different tasks. However, to use a workflow the client must also run GridNexus to use them. In many instances it may be beneficial to provide the workflow as a functioning web service. This would better facilitate workflows by adding three key benefits: security, accessibility and maintainability.

Benefit	Explanation
Security	Hide details such as paths. Prevent users from changing workflow details.
Accessibility	Make available to any Web service enabled client, .Net, Java, C++, etc.
Maintainability	If sharing workflow, create one edit point which would impact all users.

Table 5. Benefits of Publishing Workflows as Web Services

3.2 Deployment

Once complete, JXPL will generate an archive file containing the requirements to run the service. The compiled war file may either be run in a light weight J2EE container running on the local user's machine for testing, debugging or demonstration. Alternatively the archive may be deployed in a production J2EE application container making the workflow available to many users.

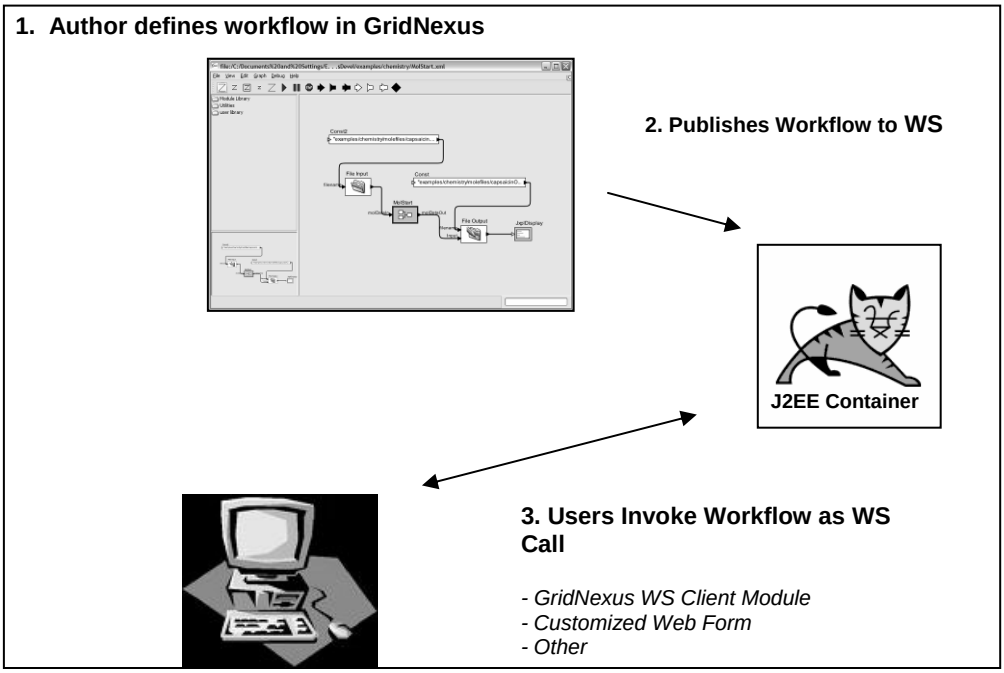


Figure 8. Workflow Publication

3.3 Methodologies

Publishing of workflows require three key components. First, a means of describing workflows as procedure calls needs to be defined. Second, a means of representing these calls with a WSDL interface needs to be described. Last, a method in which the web service container can connect the web service invocation to the JXPL Script needs to be implemented.

Rarely is there only one solution to a problem. During analysis, two potential methods were identified for publishing JXPL workflows as a web service. The methods that were explored were Wrapper Approach, and Dynamic Approach. Each method has its strengths and weaknesses. The Wrapper Approach was chosen for implementation due to its ease of implementation and because it required no modification of the Axis2 API.

3.3.1 Wrapper Approach

In the wrapper approach, a special class would be generated that would wrap all necessary functionality into a source file and then need to be compiled before it could be deployed. Although simple, this approach requires the presence of a compiler and involves auto generating and compiling new source code. This approach would place most of the effort of creating a mapping up front. The weakness of this approach is the need for dynamically compiling code. The WSDL would be generated dynamically from the wrappers template.

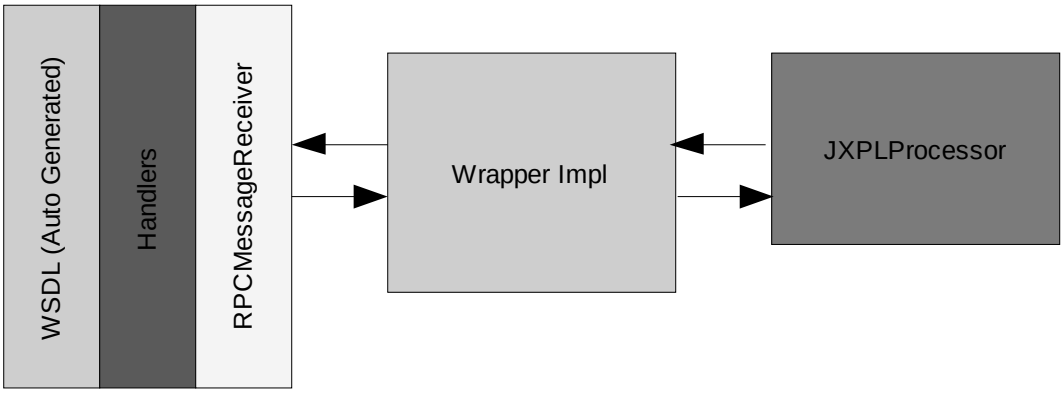


Figure 6. Wrapper Approach

3.3.2 Dynamic Approach

In the dynamic approach, no new source code needs to be generated and all can be done dynamically with precompiled code. However what needs to be generated with this approach are the WSDL documents for each prospective service. Binding in this scenario would be accomplished by custom Axis2 message receivers that will in turn map the incoming soap messages into JXPL requests. The custom receiver would sever the typical ImplClass – Message Receiver relationship to create a JXPL Processor - Message

Receiver relationship. This approach is advantageous in that the JXPL Engine need only be concerned with generating XML (as it itself is an XML language) instead of building new wrapper classes, The Axis receiver itself can then negotiate the mapping.

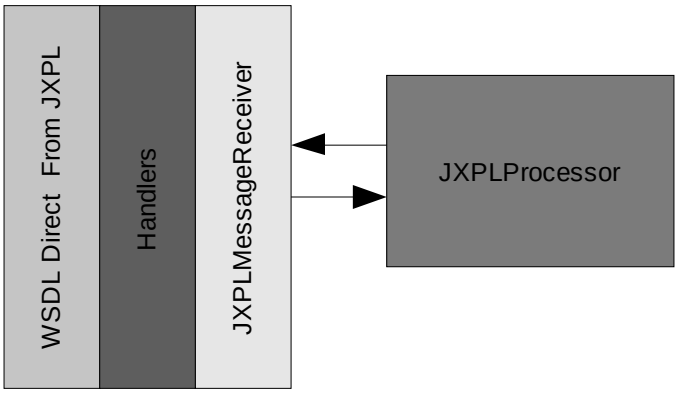


Figure 7. Fully Dynamic Approach

However, because this approach lacks an ImplClass there is no way of utilizing the auto WSDL generator included with the Axis2 api. At time of this writing the WSDL generator can only generate a WSDL only if provided a compiled Java Class. This means that a JXPL to WSDL generator would have to be created. As previously discussed, the WSDL is a very complicated document and would add unneeded complexity to the implementation.

3.4 JXPL Extensions

To describe the Web service being tested, a set of primitives have been included to describe the service, as well as the operations and parameter types that are associated with the service.

3.4.1 SOAPService

The SOAP Service actor is the primary actor that describes the SOAP Service.

Properties associated with the SOAP Service include “Name”, “Namespace”, and a toggle determining whether or not the service is going to be packaged for deployment or run locally. At runtime, if the service is to be run locally, it will block further execution of the thread until the thread is shutdown by the user. On remote run JXPL processors, this mode of operation will not permitted unless specified in the policy file. This primitive will return the JxplSymbol “true”. The ultimate task of this primitive is to ensure that the service is defined and generated properly.

3.4.2 SOAPOperation

The SOAPOperation is the primitive that provides the executable portions of the SOAP Service. JXPL already has a function called “Defun” that enables the user to define functions. SOAPOperation elaborates on the same idea as “Defun.” The difference with SOAPOperation is an expanded set of parameters that enable it to specify the “type” of parameters the operation bindings will accept and return. To make it easier to use, the service will specify parameters based upon example. A type map will convert the JXPL types to the appropriate interface types at runtime. A sample of the type mapping is illustrated in the table below.

JXPL Type	Java Type (as needed for Java2WSDL Generator)	XSD Schema Type
jxpl:integer	int	xsd:int
jxpl:decimal	double	xsd:double
jxpl:list	Object	xsd:anyType
jxpl:string	String	xsd:string
jxpl:xml	Object	xsd:anyType

Table 6. Type Mapping JXPL - Java – XSD

3.5 Example of a GridNexus Workflow Published as a Web Service

The following figure is an example of a workflow published as a simple web service. The workflow will accept a single integer and double its value. The Soap Service module accepts two inputs in the editor, a “SoapOperation” (defining business logic, parameters and return type), and a location in which to store the file. Upon execution of this script the archive will be written to the file “DoublerService.aar” which may then be deployed in an J2EE application container.

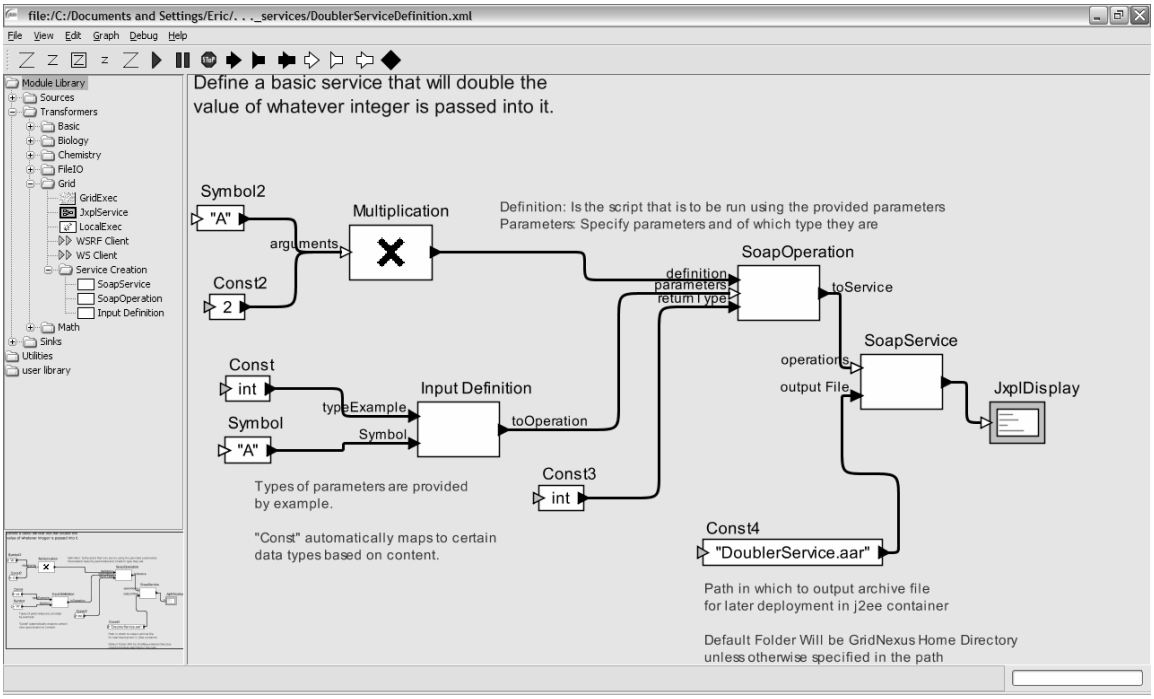


Figure 8. Gridnexus Definition of a Service That Doubles a Number Passed into it

4. Evaluation and Discussion

4.1 GridNexus to Other Web Process Editors

Now that we have illustrated the basics we will adopt a more complex orchestration process in which to compare the different editors. For this example we will explore how the different toolkits would implement a Loan Approval service that does the following

- Accept a loan amount (for illustration purposes we will just assume an integer value)
- Send the amount through the SmallLoan service to determine if the risk is high or not.
- **IF** the service decides that the risk is high, forward to the BigLoan service for a higher approval decision.
- **IF NOT**, then approve the loan.

This example would serve to illustrate two key features: (1) the ability to coordinate web services, and (2) to apply additional logic that determines a path through the process.

4.1.1 JOpera

4.1.1.1 Runtime Differences

GridNexus and JOpera are alike in goal but dissimilar in how they approach it there. JOpera contains a modeling language but for the executable runtime code relies on compiling Java byte code through the framework provided by Eclipse.

In comparison Gridnexus does consist of a modeling language borrowed from its Ptolemy front end. However at runtime, the GUI will generate an XML runtime JXPL script to facilitate the execution of the workflow, as opposed to byte code [29].

4.1.1.2 Creation Tool Differences

JOpera has many robust options for creating workflows. Many of which are based around constructs found in the Eclipse Development environment. As such many of these options require going through a set of configuration screens to get there.

By contrast, GridNexus, while not based on Eclipse, contains a set of modules are already available to perform even trivial tasks. Instead of having to go through a variety of menus, GridNexus is driven by a dialogue box accessed from double clicking on an actor. Thus eliminating the need to go through a sequence of views to establish the configuration the user wants. The only time the user need concern themselves with multiple views of a workflow is if the user opts to compartmentalize the visual workflow into composites.

In JOpera, to create web service modules, the user must go under the "New" menu to add a "WSDL" for each of the Web Services, each time running through a series of steps required to import them as program modules to be used in the workflow. After the program modules have been created, the user must then switch to the processes Dataflow graph. The user will first import each of the modules required. Second the user must expand each module to reveal its input and output components and connect them accordingly.

Because the Dataflow graph doesn't include any conditional support, JOpera utilizes a separate graph which manages Control Flow. In the Control Flow graph each module is either going to be true or false (on or off). If true (or switched to "true" by an event) the workflow will be required to execute the module upon receiving data. Otherwise, the module will be skipped.

GridNexus does not separate the Dataflow from the Control flow. Instead control is determined by a set of primitives designed for handling conditionals (behaving much like a switch statement). Another key difference is that, while JOpera creates modules by going through a set of menus, GridNexus creates them by including a pre-created library of adjustable modules for a variety of different tasks.

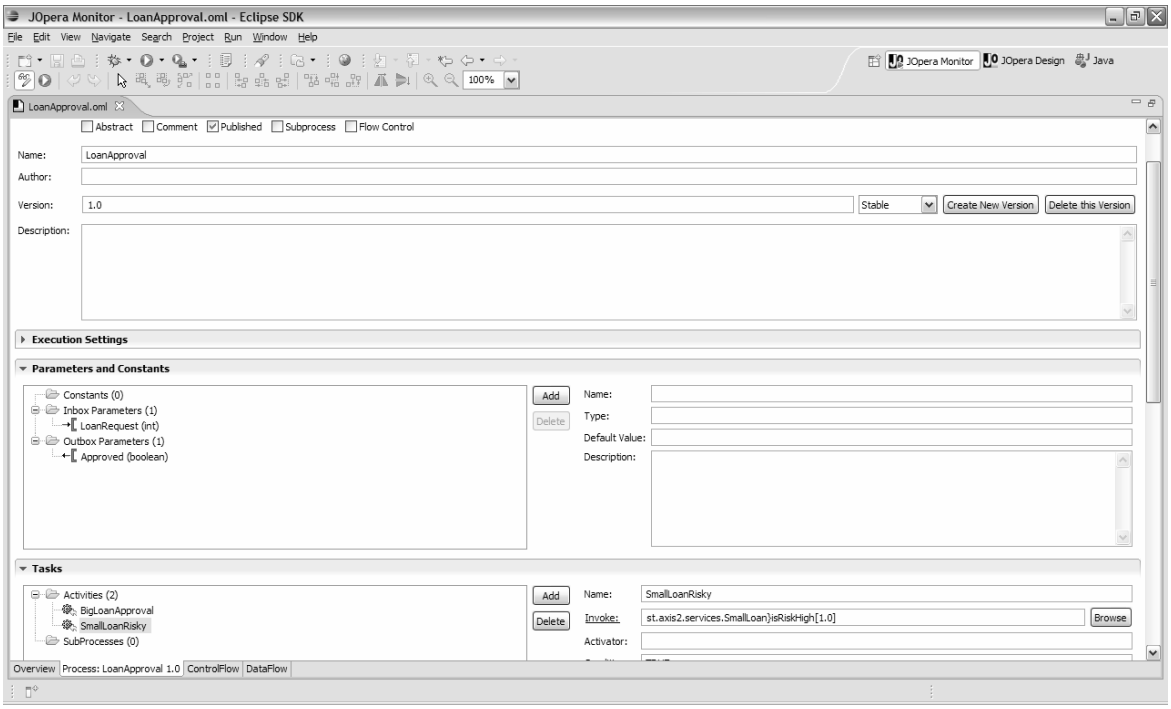


Figure 9 Loan Approval Process Overview (showing options for creating sub processes and activities)

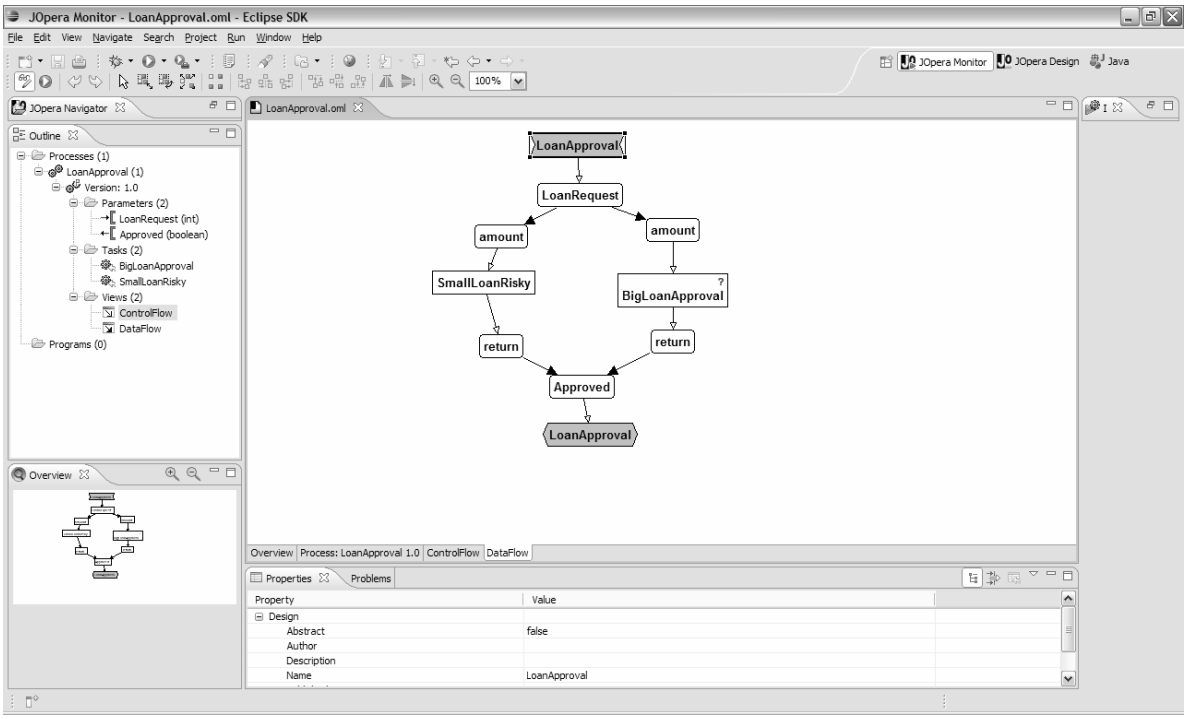


Figure 10 JOpera Data Flow for Loan Approval Example

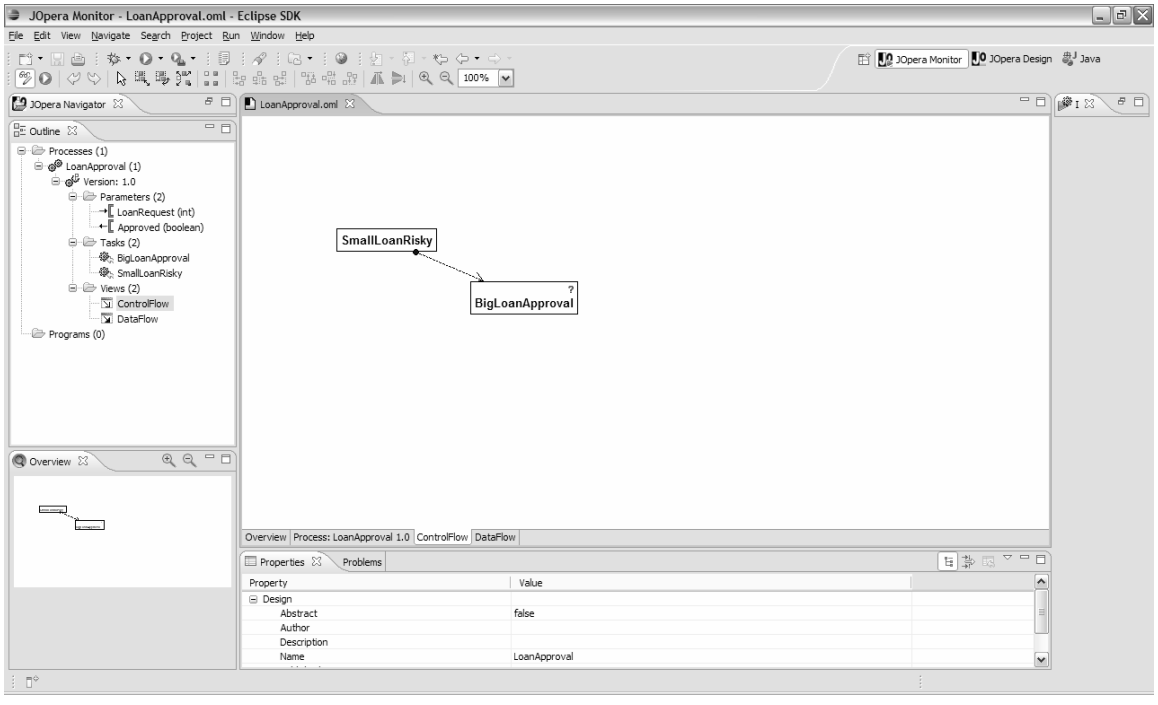


Figure 11. JOpera Control Flow

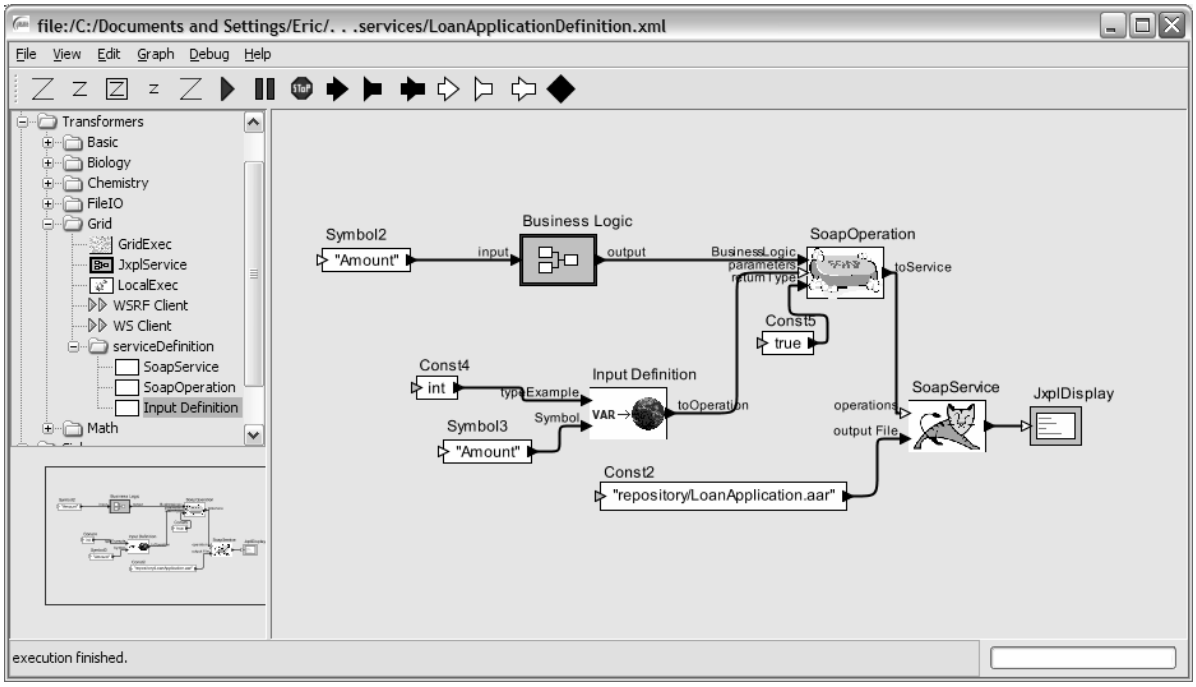


Figure 12. GridNexus Representing Loan Approver Workflow

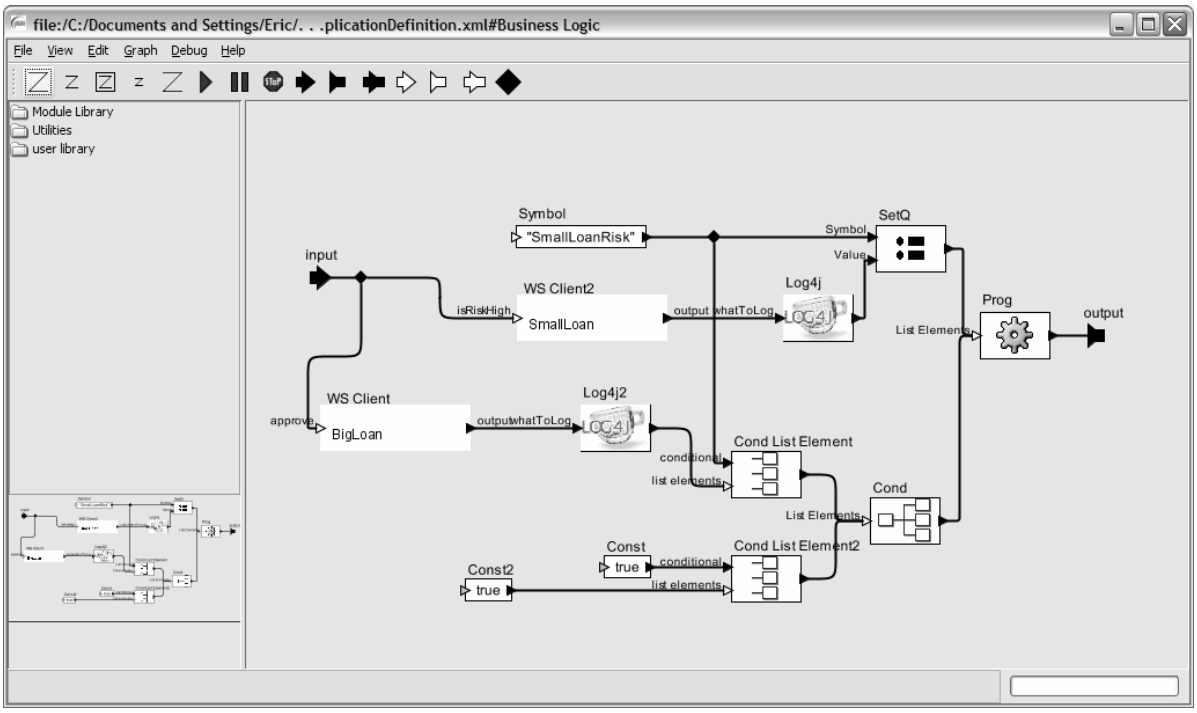


Figure 13. Internal Logic of GridNexus Process (with logging capability inserted)

4.1.2 OMII/BPEL Project for Eclipse

4.1.2.1 Creation Tool Differences

Like JOpera, OMII also takes advantage of Eclipse for its front end. However being based on the BPEL Project, contains a set of predefined modules that correspond to specific functions found within the BPEL language. Creating connections to web services involve using an import wizard to grab the WSDL's and binding them to "Partner Links" before they can be used in the web service. "Partner Links" are then used to define which web service to invoke from within the BPEL script (defined in the modules' properties).

The idea of the BPEL Project to include predefined modules is similar to that of GridNexus. For conditional support, the BPEL Project even includes a switch statement from which to make decisions and include splits in the path of the workflow. However, the BPEL Project utilizes XPath statements to evaluate logic, whereas GridNexus includes special modules for logical comparisons.

4.1.2.2 Runtime Differences

At runtime the BPEL script is deployed on the server and compiled as a web service. Evaluation is performed by a special engine that is designed to parse the BPEL XML script from Eclipse and bind it to executable byte code. Afterwards, the script can be invoked as a typical web service. GridNexus is very similar in that JXPL gets bound and interpreted as a function call. However, one difference with JXPL is that after being deployed as a web service, it is not bound to byte code until first invoked, when the JXPL Processor is first invoked. The BPEL engine is bound to byte code during deployment.

4.1.3 Microsoft WF Framework

For organizations seeking a proprietary .NET based solution, Microsoft is currently working to release their WF Framework .NET 3.0 extension for Visual Studio 2005/2007. Microsoft plans to use its workflow environment to create WS-BPEL 2.0 scripts that can be deployed within their next release of BizTalk Server.

4.1.4 Building and Consuming Web Services with Java Source and Axis2 Utilities

Axis2 provides many utilities that may help programmers build web services from source. Programmers are typically offered 2 primary options for development.

Option one is to create the WSDL first and then use the provided WSDL2Java utility to parse an interface in which the programmer will insert the corresponding programming code. The second option is to use the Plain Old Java Object (POJO) approach. In this approach the programmer need only implement the application and method calls as a normal Java class and then let the Axis2 deployment process invoke an automatic WSDL generator. In both approaches the programmer is also required to provide a description document (services.xml) that describes web service by configuring the Message Receiver, Message Exchange Pattern, and Implementation class.

```
<service name="MyProject" scope="application">
<description>Published Service</description>
<messageReceivers>
<messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
                    class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
<messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
                    class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
</messageReceivers>
<parameter name="ServiceClass" locked="false">sample.MyProject</parameter>
</service>
```

Figure 14. Services.xml File

Finally, the application must be packaged into a jar file (ending with the extension ".aar") for deployment. An example of deployment file directory structure is listed here.

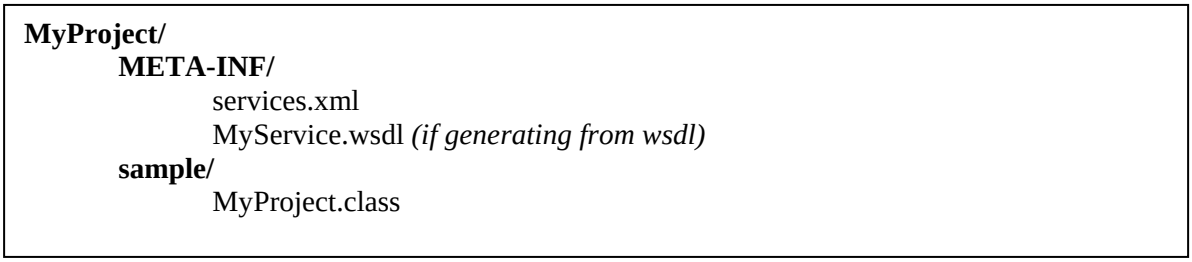


Figure 15. Structure of Deployment File

To create client applications, the programmer would again use the special WSDL2Java utility, providing a URL to the WSDL and allow the utility to parse and generate the appropriate data bindings and stubs for invoking the web service. The programmer would then use these to build an adequate client or incorporate into a larger application [24].

4.2 Scripting Comparison of BPEL to JXPL

4.2.1 Strengths of BPEL

WS-BPEL is a standard agreed upon by a committee of SOA leaders. It has been adopted by a variety of vendors including IBM, Oracle, Active Endpoints and more [19].

Another point worth mentioning is WS-BPEL's focus on function and not the visual representation. This cuts out a lot of XML bloat and enables vendors to produce their own visual front ends with their own visual formats for representing the process graphically. WS-BPEL 2.0 also provides hooks for extensibility, although engines such as active-endpoints do not yet support this feature [25].

4.2.2 Weaknesses of BPEL

One thing that BPEL cannot do is execute anything else besides web services. BPEL is a standard for coordinating WS activities. If a local application or mathematical equation needs to be invoked, these tasks must be provided as a web service by other means.

4.2.3 Strengths of JXPL

Like BPEL JXPL is also separated from its modeling language. JXPL focuses on the function of the script rather then the visual representation. At present the GridNexus GUI is the only gui application available for JXPL but others can be easily adopted as JXPL shares no tie with its GUI, much unlike JOpera.

Unlike BPEL JXPL has the ability to invoke more than just web services. JXPL has the added capability for invoking local applications, and passing portions of the workflow to remote JXPL processors for distributed processing.

If a custom set of functionality is required for a process that cannot be obtained from composing existing modules, JXPL allows the development of new modules simply by implementing a common “Primitive” interface and ensuring they belong to a subdivision of the “org.jxpl.primitives” package.

A final strength of JXPL is that the engine and editor are bundled together and offered as Open Source for adoption and extension by the Open Source community.

4.2.3 Weaknesses of JXPL

Because JXPL is inspired by LISP, it can easily become harder to write and read by hand at times, due to its layers of nested lists. Fortunately, as with many XML

languages these task are intended to be handled by machine and not by human.

5. Summary and Conclusions

Adding the ability to publish workflows as Web Services helps make GridNexus a more powerful and valuable tool. Now GridNexus can not only be used to orchestrate, but also to share web services easily and securely. Additionally, building GridNexus' WS support on the Axis2 ensures that GridNexus can easily be adapted to support many of the latest features and expansions developed for the Axis2 web services stack. Table 7 below summarizes the results of the comparisons between the different software toolkits discussed throughout the paper.

Issue	GridNexus	BPEL for Eclipse	JOpera
Extensibility	*Very Good* New primitive and add to classpath. Invoke using qualified name.	*Bad* Active Endpoints Engine does not yet implement the extension tags specified in BPEL 2.0.	*Good* Java Snippets offer limited extensibility. Ability to add new frameworks.
Portability	*Very Good* Deploy in any axis2 enabled J2EE container.	*Good* Deploy wherever Active Endpoints is installed.	*Limited* Generates its own modified web service environment. Based on older Axis stack.
Installation	*Very Good* Only requires Java SDK 1.5 or greater.	*Average* Requires Eclipse with proper dependencies.	*Moderate* Requires Eclipse with proper dependencies.
Graphical Interface Usability	*Very Good* Pre-built Drag and Drop components.	*Moderate* Pre-built drag and drop components. Eclipse can be challenging for new users and non-programmers.	*Bad* Not a true drag and drop system. Eclipse can be challenging for new users and non-programmers.

Flexibility (Multitude of Tasks)	*Very Good* Supports: Local Apps, Web Services, Grid Services, Remote Apps Logical Operations	*Bad* Only supports Web Services.	*Very Good* Supports: Local Apps, Web Services, Grid Services, Remote Apps
Robustness	*Limited* Needs support for complex types.	*Good* Targeted for industry use.	*Good*

Table 7. Comparison of GridNexus BPEL Editor for Eclipse and JOpera

Upon completion of the project, a variety of extensions were introduced into JXPL for creating Web Services directly from an XML script. To accompany these changes three GUI modules were created to enable a simplistic approach for defining a service. The *Soap Service, Soap Operation* and *Input Definition* components provide a framework to make service creation almost trivial for the user. Furthermore, the framework is extensible as it can quickly take advantage of any add-on provided for Axis2 with little to no modification necessary. In addition to these changes, it was necessary to also overhaul the original Generic WS Client in GridNexus to support the newer Axis2 web services stack, where it had originally supported Axis 1.x. Finally, all the necessary documentation and another version of GridNexus (version 2.01) was produced and published as Open Source.

6. Bibliography

1. Barry & Associates (2000-2007). Service Oriented Architecture Definition. [WWW Online]. URL http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html.

2. Hewlett-Packard Development Company (2007). Why Does SOA Matter [WWW Online]. URL http://www.systinet.com/soa_explained/why_soa_matters.

3. World Wide Web Consortium (Jan 2007). Extensible Markup Language (XML). [WWW Online]. URL http://www.w3c.org/XML.

4. World Wide Web Consortium (Jan 2007). XML Schema. [WWW Online]. URL http://www.w3c.org/XML/Schema.

5. Shodjai, P. (Jun 2006). Web services and the Microsoft Platform. [WWW Online]. URL http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnwebsrv/html/wsmsplatform.asp#wsmsplat_topic2.

6. Costello, R. (n.d.). Building Web services the REST Way [WWW Online]. URL http://www.xfront.com/REST-Web-Services.html.

7. LaMonica, Martin (2004). CNET. Where's the Simplicity. [WWW Online]. URL http://news.com.com/Wheres+the+simplicity+in+Web+services/2100-7345_3-5395630.html

8. OASIS Open (2006). SOAP. [WWW Online]. URL http://www.xml.org/xml/resources_focus_soap.shtml

9. Refsnes Data (1999-2007). SOAP Header Element. [WWW Online]. URL http://www.w3schools.com/SOAP/soap_header.asp

10. OASIS Open (2006). WSDL. [WWW Online]. URL http://www.xml.org/xml/resources_focus_wsdl.shtml

11. Refsnes Data (1999-2007). UDDI. [WWW Online]. URL http://www.w3schools.com/SOAP/soap_header.asp

12. Thatte, S. (May 2003). Business Process Execution Language for Web services Version 1.1. [WWW Online]. URL ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf. May 2003.

13. Wikipedia. (26 Jan 2007). List of BPEL Engines [WWW Online]. URL http://en.wikipedia.org/wiki/List_of_BPEL_engines.

14. Brown J., Ferner, C., Hunt, C. JXPL: An XML-based Scripting Language for Workflow Execution in a Grid Environment. [Online Serial]. URL http://people.uncw.edu/cferner/papers/NDS_47235140_5.pdf.

15. Hudson, T., Stapleton, A., Brown J. Codifying Bioinformatics Processes Without Programming (July 2004). BioSilico. Vol 2. No 4. [Online Serial]. http://www.gridnexus.org/_files/Hudson-Biosilico-2-4-Final.pdf.

16. EECS UC Berkeley (2006). Ptolemy II Frequently Asked Questions [WWW Online]. URL http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIfaq.htm.

17. UNCW. (Nov 2006). UNCW Grid Computing Project. [WWW Online]. URL http://gridnexus.org/index0.php?session=overview.

18. Brown J. et al (Jun 2005). GridNexus: A Grid Services Scientific Workflow System International Journal of Computer Information Science (IJCIS), Vol 6, No 2. p72-82. [Online Serial]. http://gridnexus.org/_files/IJCIS2005_GridNexus.pdf.

19. Jorden, D. et al (Apr 2007). Web Services Business Process Execution Language Version 2.0. Oasis. [Online Serial]. URL http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

20. Erl, Thomas. Pearson Education, Inc. (2004). Upper Saddle River, New Jersey 07458. Service-Oriented Architecture. [BOOK]. ISBN 0-13-142898-5.

21. ETH Zurich. (November 2005). JOpera: Process Support for Web Services. [WWW Online]. URL http://www.iks.ethz.ch/jopera.

22. Heinis, T. JOpera Overview (July 2006). [Online Serial]. URL http://www.jopera.ethz.ch/docs/jopera_overview.pdf.

23. Apache Foundation. Axis2 User's Guide -Building Services. [WWW Online]. URL http://ws.apache.org/axis2/1_2/pojoguide.html

24. Apache Foundation. Axis2 User's Guide -Creating Clients. [WWW Online]. URL http://ws.apache.org/axis2/1_2/userguide-creatingclients.html

25. Active Endpoints. Active BPEL Open Source Project (2007). [WWW Online]. URL http://www.active-endpoints.com/active-bpel-engine-overview.htm

26. O'Reilly, Tim. O'Reilly xml.com (2003). REST vs. SOAP at Amazon. [WWW Online]. URL http://www.oreillynet.com/pub/wlg/3005.

27. Heath B., Vetter R. University North Carolina Wilmington and East Main Consulting (2007). Fostering Undergraduate Research Partnerships through a Graphical User Environment for the North Carolina Computing Grid: Final Report. [Online Serial]. http://www.gridnexus.org

28. Wilkinson B., Ferner C. Teaching Grid Computing Across North Carolina Part II IEEE Distributed Systems Online, Vol 7, No 7; July 2006. [Online Serial]. URL http://www.gridnexus.org/_files/o7003.pdf.

29. ETH Zurich. JOpera Manual (June 2007). [WWW Online]. URL http://www.iks.inf.ethz.ch/jopera/doc/jop.html/jop_2.html#2.4.3

7. Appendix

7.1 Loan Approval Script

7.1.1 JXPL Script

```
<jxpl:list xmlns:jxpl="http://www.jxpl.org/script">
  <jxpl:primitive name=".service.soap.SOAPService">
    <jxpl:list>
      <jxpl:primitive name="Property"/>
      <jxpl:string value="name"/>
      <jxpl:string value="MyService"/>
    </jxpl:list>
    <jxpl:list>
      <jxpl:primitive name="Property"/>
      <jxpl:string value="namespace"/>
      <jxpl:string value="http://jxpl.org/service"/>
    </jxpl:list>
    <jxpl:list>
      <jxpl:primitive name="Property"/>
      <jxpl:string value="package"/>
      <jxpl:symbol name="true"/>
    </jxpl:list>
    <jxpl:list>
      <jxpl:primitive name="Property"/>
      <jxpl:string value="generate"/>
      <jxpl:symbol name="true"/>
    </jxpl:list>
    <jxpl:list>
      <jxpl:primitive name="Property"/>
      <jxpl:string value="outputFile"/>
      <jxpl:string value="repository/LoanApplication.aar"/>
    </jxpl:list>
  </jxpl:primitive>
  <jxpl:list><!-- List of Functions -->
    <jxpl:list><!-- Function (name: amIApproved) -->
      <jxpl:primitive name=".service.soap.SOAPOperation">
        <jxpl:list>
          <jxpl:primitive name="Property"/>
          <jxpl:string value="name"/>
          <jxpl:string value="amIApproved"/>
        </jxpl:list>
        <jxpl:list>
          <jxpl:primitive name="Property"/>
          <jxpl:string value="return"/>
          <jxpl:symbol name="true"/>
        </jxpl:list>
      </jxpl:primitive>
      <jxpl:list><!-- Function Definition  -->
        <jxpl:primitive name="Prog"/>
        <jxpl:list>
          <jxpl:primitive name="SetQ"/>
          <jxpl:symbol name="SmallLoanRisk"/>
           <jxpl:list><!-- Invoke Smaller Service -->
```

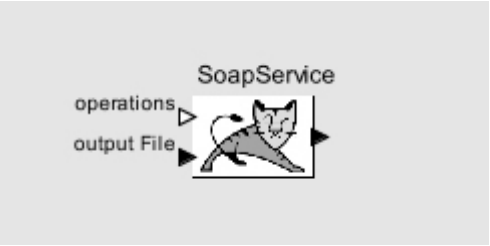
```
            <jxpl:primitive name=".ws.WSClient">
              <jxpl:property name="WS-Security">
                <jxpl:symbol name="true"/>
              </jxpl:property>
            </jxpl:primitive>
            <jxpl:string value="http://localhost:8080/axis2/services/SmallLoan?wsdl"/>
            <jxpl:string value="http://localhost:8080/axis2/services/SmallLoan"/>
            <jxpl:string value="isRiskHigh"/>
            <jxpl:symbol name="Amount"/>
          </jxpl:list>
        </jxpl:list>
        <jxpl:list>
          <jxpl:primitive name="Cond"/>
          <jxpl:list>
            <jxpl:symbol name="SmallLoanRisk"/>
            <jxpl:list><!-- Invoke BigLoan Service -->
              <jxpl:primitive name=".ws.WSClient">
                <jxpl:property name="WS-Security">
                  <jxpl:symbol name="true"/>
                </jxpl:property>
              </jxpl:primitive>
              <jxpl:string value="http://localhost:8080/axis2/services/BigLoan?wsdl"/>
              <jxpl:string value="http://localhost:8080/axis2/services/BigLoan"/>
              <jxpl:string value="approve"/>
              <jxpl:symbol name="Amount"/>
            </jxpl:list>
          </jxpl:list>
          <jxpl:list>
            <jxpl:symbol name="true"/>
            <jxpl:symbol name="true"/>
          </jxpl:list>
        </jxpl:list>
      </jxpl:list>
      <jxpl:list><!-- Parameters for (amIApproved) -->
        <jxpl:list><!-- param0 -->
          <jxpl:integer value="0"/><!-- Type Example -->
          <jxpl:symbol name="Amount"/><!-- variable -->
        </jxpl:list>
      </jxpl:list>
    </jxpl:list>
  </jxpl:list>
</jxpl:list>
```

7.1.2 BPEL Script

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<bpws:process xmlns:bpws="http://schemas.xmlsoap.org/ws/2004/03/business-process/"
xmlns:ns="http://Eclipse.org/bpel/sampleArtifacts" xmlns:ns0="http://bank"
xmlns:ns1="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://Eclipse.org/bpel/sample"
exitOnStandardFault="yes" name="LoanApproval" suppressJoinFailure="yes"
targetNamespace="http://Eclipse.org/bpel/sample">
  <bpws:import importType="http://schemas.xmlsoap.org/wsdl/" location="LoanApproval.wsdl"
namespace="http://Eclipse.org/bpel/sample"/>
  <bpws:import importType="http://schemas.xmlsoap.org/wsdl/" location="file://SmallLoan.wsdl"
namespace="http://bank"/>
  <bpws:import importType="http://schemas.xmlsoap.org/wsdl/"
namespace="http://Eclipse.org/bpel/sampleArtifacts"/>
  <bpws:import importType="http://schemas.xmlsoap.org/wsdl/" location="file://BigLoan.wsdl"
namespace="http://bank"/>
  <bpws:import importType="http://www.w3.org/2001/XMLSchema"
location="bundleentry://632/cache/www.w3.org/2001/XMLSchema.xsd"
namespace="http://www.w3.org/2001/XMLSchema"/>
  <bpws:import importType="http://schemas.xmlsoap.org/wsdl/" location="LoanApprovalArtifacts.wsdl"
namespace="http://Eclipse.org/bpel/sampleArtifacts"/>
  <bpws:import importType="http://schemas.xmlsoap.org/wsdl/" location="BigLoan.wsdl"
namespace="http://bank"/>
  <bpws:import importType="http://schemas.xmlsoap.org/wsdl/" location="SmallLoan.wsdl"
namespace="http://bank"/>
  <bpws:partnerLinks>
    <bpws:partnerLink myRole="LoanApprovalProvider" name="client" partnerLinkType="tns:LoanApproval"/>
    <bpws:partnerLink name="BigLoan" partnerLinkType="ns:BigLoan" partnerRole="BigLoanApprover"/>
    <bpws:partnerLink name="SmallLoan" partnerLinkType="ns:SmallLoan"
partnerRole="SmallLoanRiskAppraiser"/>
  </bpws:partnerLinks>
  <bpws:variables>
    <bpws:variable messageType="tns:LoanApprovalRequestMessage" name="input"/>
    <bpws:variable messageType="tns:LoanApprovalResponseMessage" name="output"/>
    <bpws:variable messageType="ns0:isRiskHighMessage" name="InputVariable"/>
    <bpws:variable name="loanAmount" type="ns1:int"/>
    <bpws:variable name="approved" type="ns1:boolean"/>
    <bpws:variable messageType="ns0:isRiskHighResponse" name="SmallLoanResponse"/>
    <bpws:variable messageType="ns0:isRiskHighMessage" name="SmallLoanRequest"/>
  </bpws:variables>
  <bpws:sequence name="main">
    <bpws:receive createInstance="yes" name="receiveInput" operation="process" partnerLink="client"
portType="tns:LoanApproval" variable="input"/>
    <bpws:assign name="Assign" validate="yes">
      <bpws:copy>
        <bpws:from part="payload" variable="input">
          <bpws:query queryLanguage="http://www.w3.org/TR/1999/REC-xpath-
19991116"><![CDATA[/tns:input]]></bpws:query>
        </bpws:from>
        <bpws:to variable="loanAmount"/>
      </bpws:copy>
    </bpws:assign>
    <bpws:switch name="Switch">
      <bpws:case>
        <bpws:condition><![CDATA[true(loanAmount<10000)]]></bpws:condition>
        <bpws:sequence name="HiddenSequence">
          <bpws:invoke name="InvokeSmallLoan" operation="isRiskHigh" partnerLink="SmallLoan"
portType="ns0:SmallLoanPortType"/>
          <bpws:switch name="Switch1">
```

```
          <bpws:case>
            <bpws:condition><![CDATA[true(output)]]></bpws:condition>
            <bpws:invoke name="InvokeBigLoan1" operation="approve" partnerLink="BigLoan"
portType="ns0:BigLoanPortType"/>
          </bpws:case>
        </bpws:switch>
      </bpws:sequence>
    </bpws:case>
    <bpws:otherwise>
      <bpws:invoke inputVariable="SmallLoanRequest" name="InvokeBigLoan" operation="isRiskHigh"
outputVariable="SmallLoanResponse" partnerLink="SmallLoan" portType="ns0:BigLoanPortType"/>
    </bpws:otherwise>
  </bpws:switch>
  <bpws:reply name="replyOutput" operation="process" partnerLink="client" portType="tns:LoanApproval"
variable="output"/>
 </bpws:sequence>
</bpws:process>
```

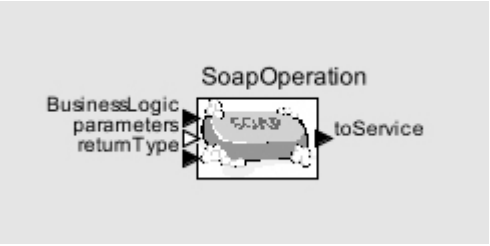
7.2 GridNexus Service Definition Modules



Purpose: Defines the service.

Inputs: Soap Operations for the service, location for the packaged archive file.

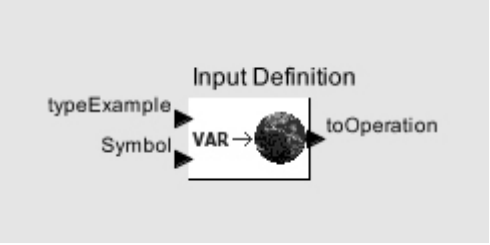
Output: Deployable Archive File for the application *(i.e. "<name>.aar").*



Purpose: Defines an operation of the service.

Inputs: Workflow to be used for Business Logic, Input Definitions, and Return Type

Output: Operation definition for use with SoapService.



Purpose: Defines variable name and type for use in SoapOperation

Inputs: Symbol representing variable and a const providing a type example.

Output: Parameter for use with SoapOperation.

7.3 Visually Defining a Web Service in GridNexus

Objective:
Define a Web Service

Description:
This tutorial will illustrate how to build web services from workflows. To illustrate, we will be building a service that multiplies two numbers and deploy it in a j2ee container.

Prerequisites:
We assume that you have or have access to a web server with Axis2 already installed and running. Additionally, we assume that you have an SDK of Java installed. Also ensure that the following jar files are included in the classpath for Axis2 on the container machine. They can be found in the GridNexus/lib folder.

- jxpl.jar
- jxpl-ws.jar

.

Step By Step Instructions:

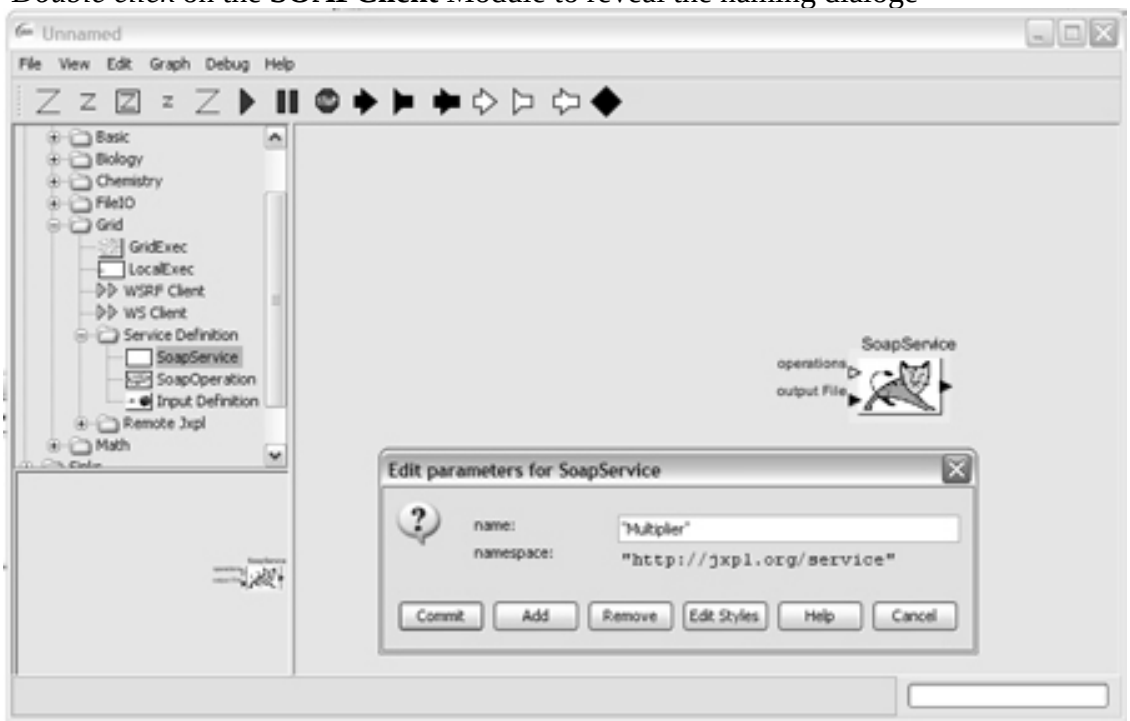
Open **GridNexus** on your workstation/personal computer

Select **File->New->Workflow**

In the folders on the left *expand* **Module Library->Transformers->Grid->Service Definition Folder**

Drag an instance of **SOAPService** into the Workspace.

Double click on the **SOAPClient** Module to reveal the naming dialoge



In the Name box *enter* the **Multiplier**

Now that we have named the service, we will now begin defining the operations. *Drag* a **SOAPOperation** into the workspace.

Double click the **SoapOperation** module and specify the name "Multiply."

Connect the **SoapOperation** to the "operations" input on the **SoapService** module

Now we need to start defining parameters. *Drag* a **Input Definition** into the workspace. Here you should take type into account. The scheme for declaration here is "type by example" as seen in the table below. For declaring types we will need to *drag* both a **Sources->const and Sources->symbol** into the workspace. The **symbol** will represent the variable name and the **const** will represent type. Assign the **const** a value of *0* for integer. Next, give the symbol the name "x" in quotation marks. *Connect* the **const and symbol** to the appropiate inputs on the **Input Definition**.

Types for use with Service Definition Parameters	
Type	**Example**
integer	0
decimal	0.0
string	""
boolean	true

Repeat for another **Input Definition** with the variable name "Y"

To specify the return type *insert* another **const** with the value *0* and connect it to the returnType input on the **SoapOperation** module.

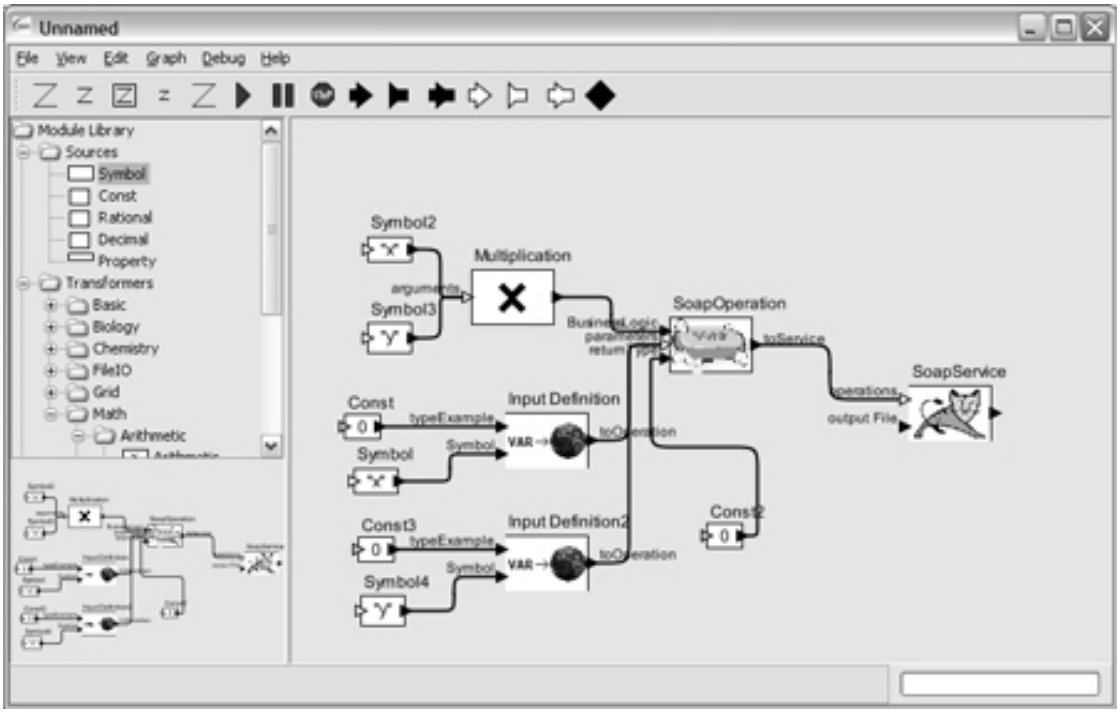
Now we must define the "Business Logic" for this operation.

From the **Modules->Math->Multiplication** menu *drag* a **Multiplication** module into the workspace.

Attach the **Multiplication** module to the "Business Logic" input on the **SoapOperation** module.

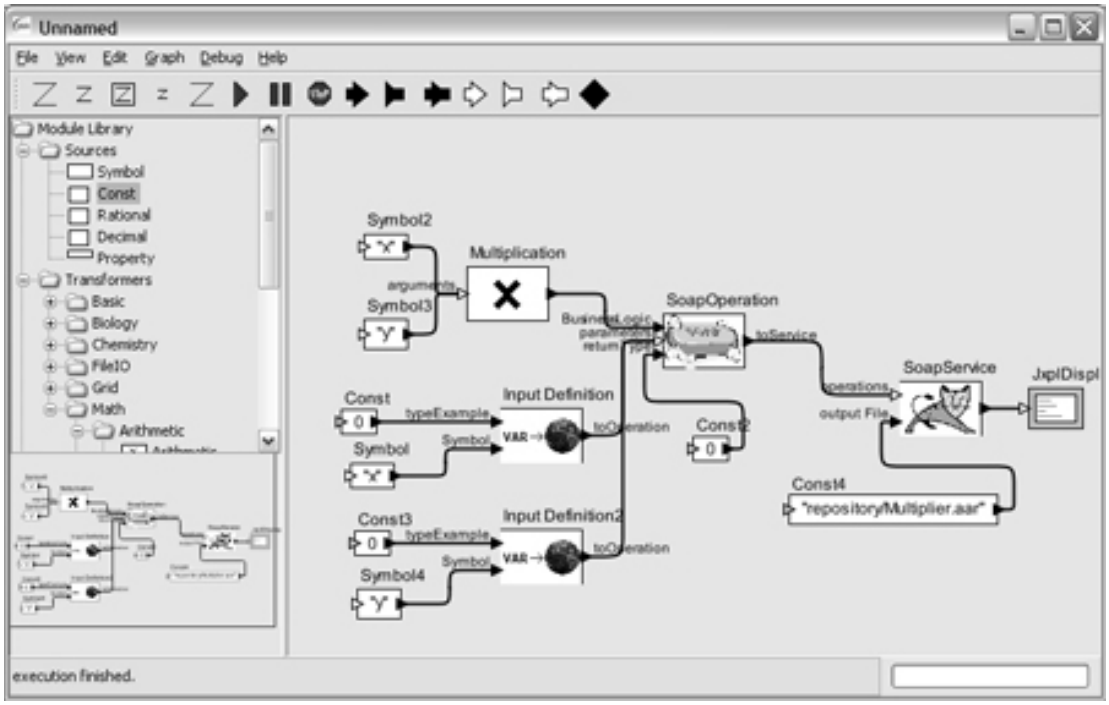
Introduce two new **Symbol** modules and give the the values "x" and "y" respectively. These will be connected to represent how the variables are to be used upon execution.

Connect both **Symbols** to the **Multiplication** module.

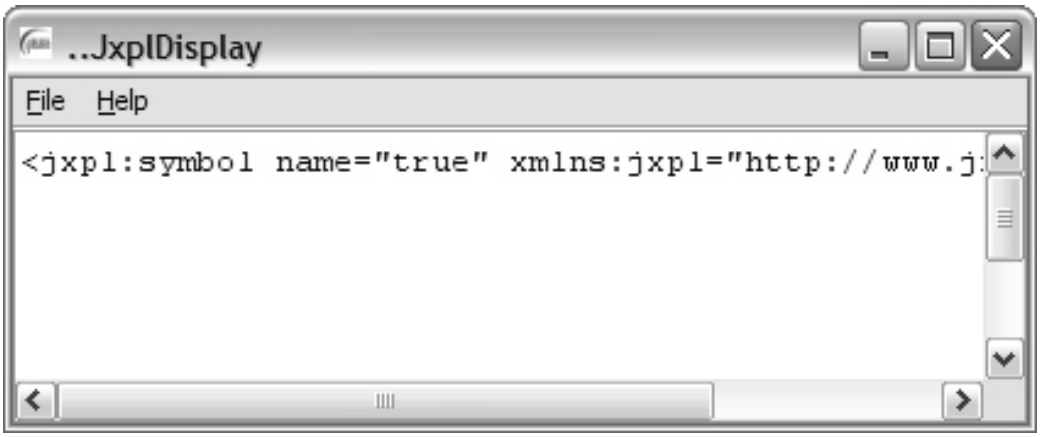


Now we are ready to package the server. *Drag* a **const** box with the value *"repository/Multiplier.aar"* and connect it to the "output file" input on the **SoapService** module.

Lastly *drag* a **JxplDisplay** into the workspace and *connect* the **SoapService** to it.

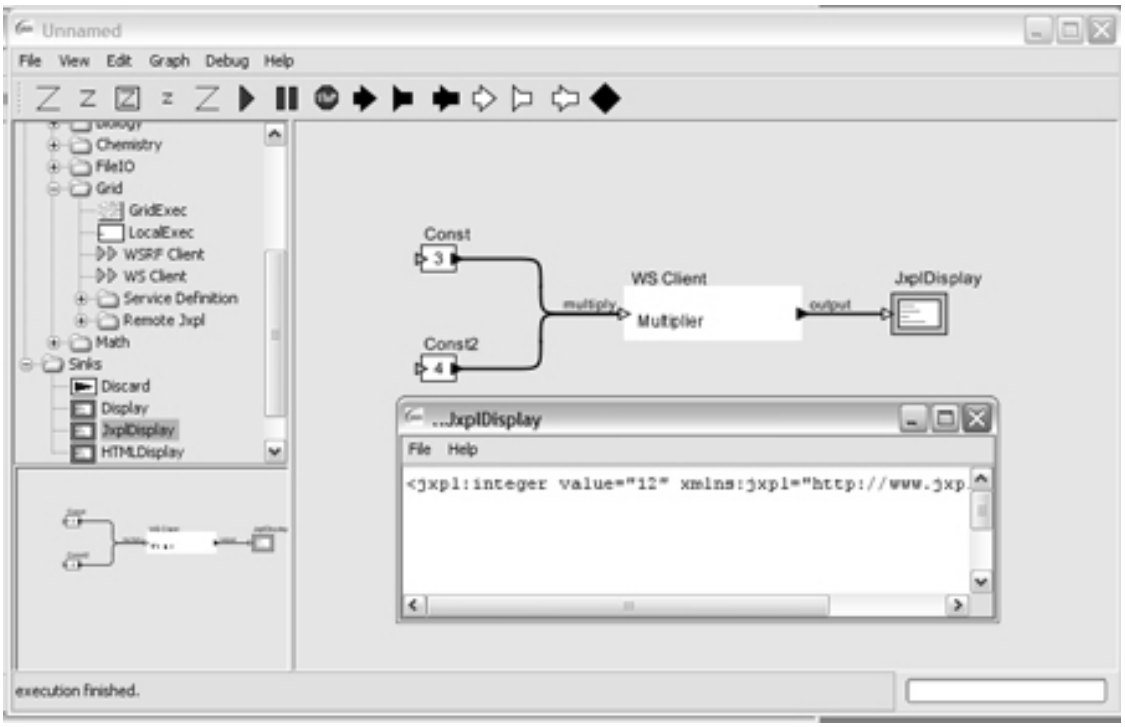


Finally *select* the **play** button (red triangle) to build the service. If you receive "true" then you have successfully created the service.



You may now deploy the service defined in your J2EE container with the aar file generated in the GridNexus/repository directory.

To test, feel free to create a WSClient (as below)...



7.4 Invoking a Web Service in GridNexus

Objective:
Invoke a Web Service

Description:
This tutorial will walk you through how to use the WS-Client to invoke the sample Version service installed by default with Axis2.

Prerequisites:
We assume that you have or have access to a web server with Axis2 already installed and running.

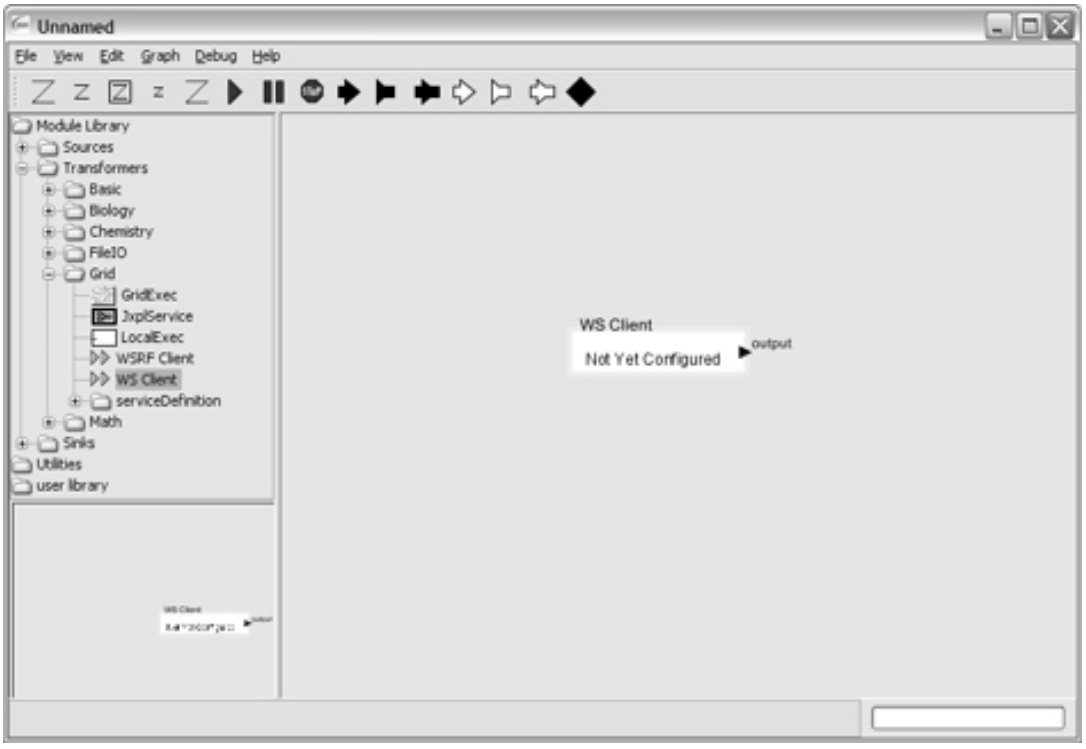
Step By Step Instructions:

Open **GridNexus** on your workstation/personal computer

Select **File->New->Workflow**

In the folders on the left *expand* **Module Library->Transformers->Grid Folder**

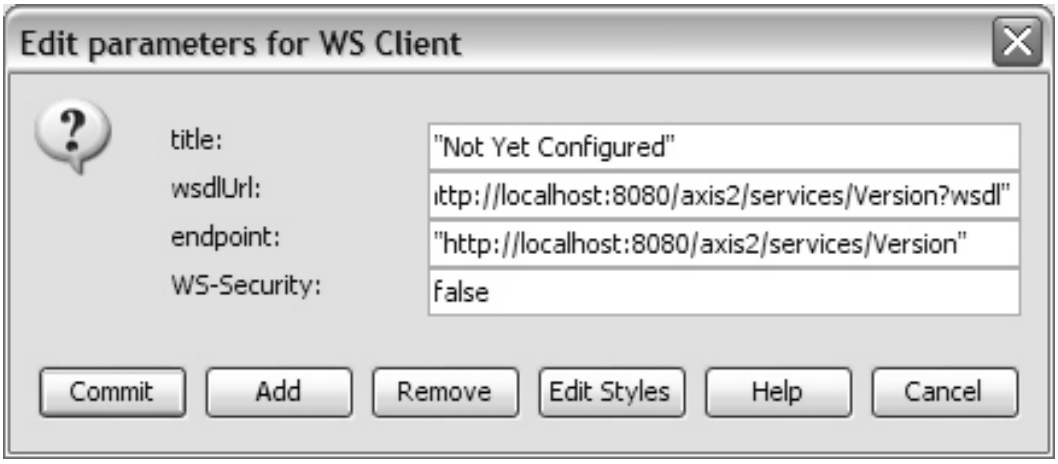
Drag an instance of **WSClient** into the Workspace



Double click on the **WSClient** Module to reveal its options

In the WSDL box *enter* the **WSDL URL** *(i.e. http://localhost:8080/axis2/services/Version?wsdl)*

In the Endpoint box *enter* the *Endpoint URL* of the web service *(i.e. http://localhost:8080/axis2/services/Version)*

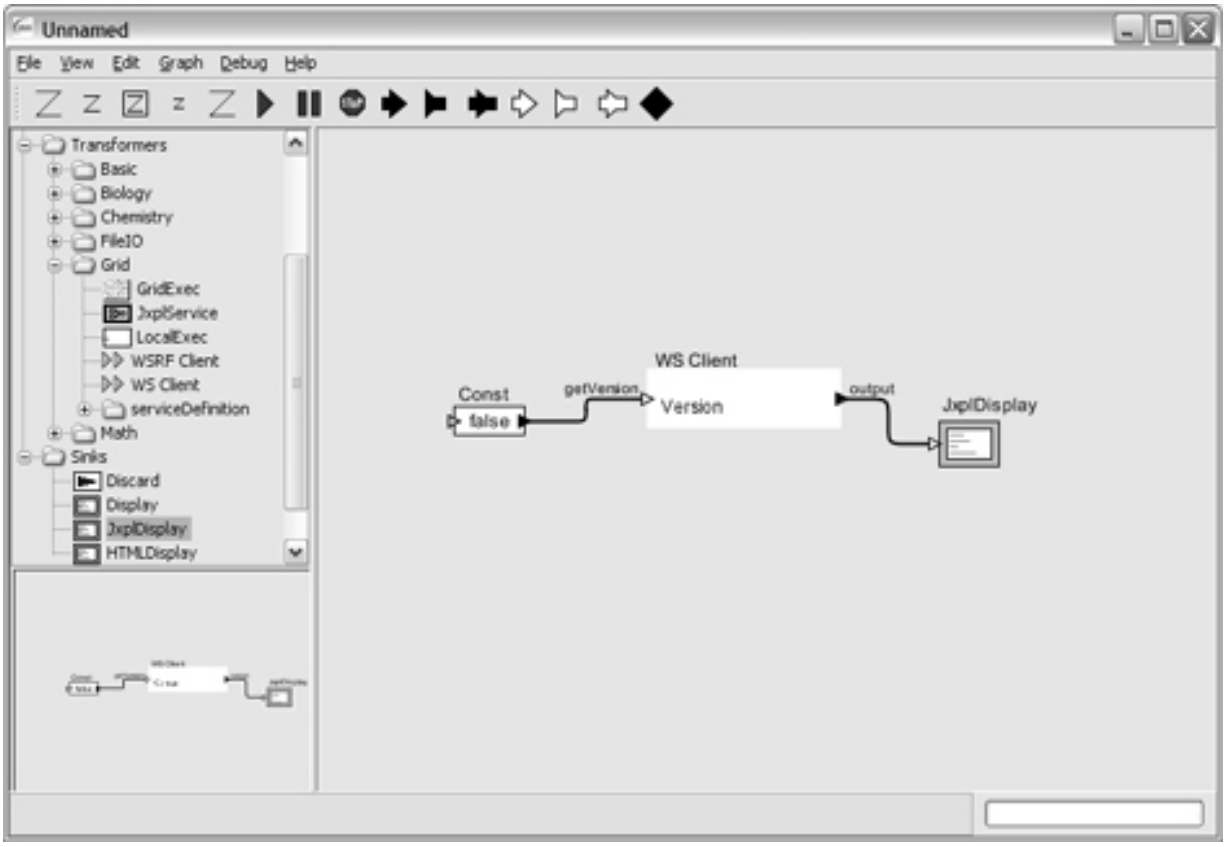


After completing this step the module should now change to reveal ports representing the available function calls

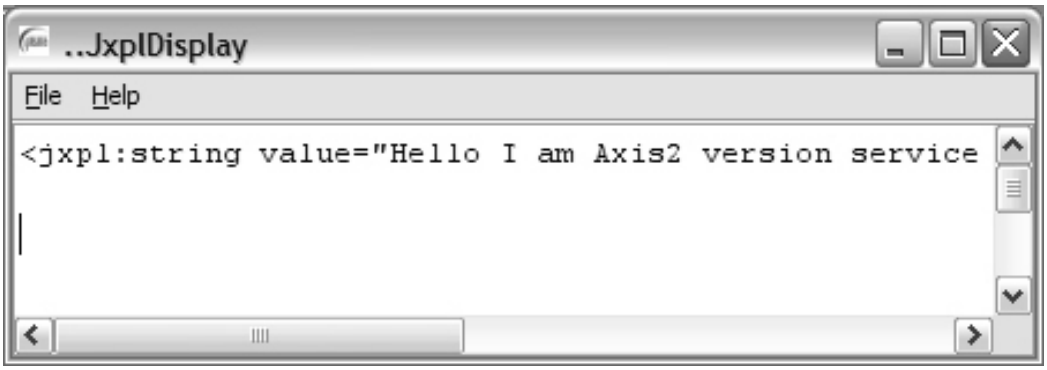
The Version service has a single function called getVersion that accepts no arguments. To select this function *drag* a **Const** from the Module Library->sources folder.

Double Click on the **Const** box and set the value to *false* (without quotations) this will serve to represent a blank parameter.

Finally *Drag* a **JxplDisplay** into the workspace from **Module Library->Sinks**



To execute the workflow *press* the **play** *(red triangle)* button. If the WSClient was configured properly and the service is available, you should get output similar to that shown below.



7.5 JXPL Source Code for Web Service Support

```
/*

Copyright © 2004-2007 UNCW. All rights reserved.

Permission is hereby granted, without written agreement and without license or
royalty fees, to use, copy, modify, and distribute this software and its
documentation for any purpose, provided that the above copyright notice and the
following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL UNCW BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT,
SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF
THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF UNCW HAS BEEN ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.

UNCW SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS"
BASIS, AND UNCW HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT,
UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
*/
package org.jxpl.primitives.service;

import org.w3c.dom.*;
import org.jxpl.*;
import org.jxpl.bindings.*;
import org.jxpl.exception.*;

import java.util.List;
import java.util.LinkedList;
import java.util.Hashtable;
import java.util.Map;
import java.net.URL;


import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
```

```
import javax.xml.namespace.QName;
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import javax.xml.parsers.*;
import javax.xml.soap.*;
import org.jxpl.exception.*;
import org.apache.commons.logging.*;

//Axis2
import org.apache.axis2.description.*;

/**
 *This class is designed to produce a Operation call from a service
 *TODO: Create construct for defining complexType from example?
 *@author Eric Harris
 *@version 0.1
 */
public class Operation implements org.jxpl.Primitive{

    private Processor environment;

    private JxplElement lambdaScript;

    private String name;

    private JxplElement returnType=null;

    private List<Param> parameters=new LinkedList();

    private static Log recorder=LogFactory.getLog(Operation.class.getName());

    public void setProcessor(Processor env){
        environment = env;
    }

    /**
    *   This cannot be run on its own...must depend on the child primitive
    *   This method will only echo the jxpl element definition
    */
    public JxplElement evaluate(JxplElement input) throws JxplException{
        //throw new JxplException("Operatin only can be used as a Child of the SOAPService Primitive");
        recorder.warn("Operation MUST be used as a child of SOAPService");
        return input;
    }

    /**
    *   Supports the Service primitive
    *   @param input    Operation Definition
    *   @param environment  JxplProcessor being utilized
    *   @return Operation containing object representation of the operation
    */
    public static Operation buildOperation(JxplElement input, Processor environment) throws JxplException{
        Operation operation=new Operation();
        operation.setProcessor(environment);
        operation.buildOperation(input);
        return operation;
    }
```

```
/**
*   This is a special utility for the Service primitive...will return a Operation Object
*   @return Operation containing Object representation of the operation
*/
public void buildOperation(JxplElement input) throws JxplException {

   List<JxplElement> list = ((JxplList)input).getElements();
   JxplPrimitive prim = (JxplPrimitive)list.get(0);
   List propList = prim.getProperties();

   Hashtable props = PropertyHelper.hash(propList, environment);


   checkProperties(props);//for extensibility

   setLambda(list.get(1));

   //get the parameter list
   List<JxplElement> paramList=((JxplList)environment.evaluate(list.get(2))).getElements();

   //get the parameters if any
   recorder.debug("Total Inputs are "+input);
   for(int i=0; i<paramList.size(); i++){
      recorder.debug("Input for parameter is: "+(JxplElement)paramList.get(i));
      Param operationParam=Param.buildParam((JxplElement)paramList.get(i), environment);
      addParameter(operationParam);
   }

}

protected void checkProperties(Map props)throws JxplPropertyException{
   if ( ! props.containsKey("name")){
      recorder.error("Missing Operation Name");
      throw new JxplPropertyException("Operation: must include a property 'name'");
   }
   else{//set the name of the operation
      setName((String)props.get("name"));
   }

   if (props.containsKey("return")){//set the return type...assume it is void otherwise
      try{
         setReturnType((JxplElement)props.get("return"));
      }
      catch(JxplException je){
         throw new JxplPropertyException("Cannot Evaluate Return Type");
      }
   }
}

/**
*   Set the name of the operation
*/
public void setName(String name){
   this.name=name;
}

/**
```

```
*   Set the script that is to be executed
*/
public void setLambda(JxplElement elem){
   lambdaScript=elem;
}

/**
*  Add Parameter to the operation
*/
public void addParameter(Param obj){

   parameters.add(obj);
}

/**
*  Set the return type (by example) to establish proper interface mapping
*  @param elem example of result type
*/
public void setReturnType(JxplElement elem)throws JxplException{
   returnType=environment.evaluate(elem);
}

/**
*
*
*/
public List<Param> getParameters(){
   return parameters;
}

public String getName(){
   return name;
}


/**
*   Get the sample of the return type
*/
public JxplElement getReturnType(){
   return returnType;
}

/**
*   Caches the operation by defining using the Defun function
*/
public void cacheOperation() throws JxplException{
   JxplList prm=new JxplList();
   //build list for parameters
   for(int i=0; i<parameters.size(); i++){
      Param p=parameters.get(i);
      prm.addElement(p.getVariable());
   }

   //build a defun function
   JxplList macroDefinition=new JxplList();
   JxplPrimitive prim=new JxplPrimitive("Defun");
   macroDefinition.addElement(prim);
   JxplSymbol macroName=new JxplSymbol(new QName(name));
   macroDefinition.addElement(macroName);
```

```
        macroDefinition.addElement(prm);
        macroDefinition.addElement(lambdaScript);
        environment.evaluate(macroDefinition);
    }
}
```

/*
Copyright © 2004-2007 UNCW. All rights reserved.

Permission is hereby granted, without written agreement and without license or
royalty fees, to use, copy, modify, and distribute this software and its
documentation for any purpose, provided that the above copyright notice and the
following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL UNCW BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT,
SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF
THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF UNCW HAS BEEN ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.

UNCW SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS"
BASIS, AND UNCW HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT,
UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
*/
```
package org.jxpl.primitives.service;

import org.w3c.dom.*;
import org.jxpl.*;
import org.jxpl.bindings.*;
import org.jxpl.exception.*;

import java.util.List;
import java.util.Hashtable;
import org.apache.log4j.*;
import java.net.URL;
/**
*   Interface for implementing a service rpc call
*/

public interface Service extends org.jxpl.Primitive{


   /**
   *   Set the processor for the sourcecode
   */
   public void setProcessor(Processor env);

   /**
   *   Evaluate the service script
   */
   public JxplElement evaluate(JxplElement input) throws JxplException;

   /**
   *   Add a new operation to the script
   */
   public void addOperation(Operation meth);

   /**
   *   get a list of operations from the script
   */
   public List getOperations();

   /**
   *  get the service name
```

```
    */
    public String getServiceName();

    /**
    *   Store script for retrieval from a wrapper
    */
    public void setScript(JxplElement script);

    /**
    *  Get the stored script
    */
    public JxplElement getScript();
}
```

/*
Copyright © 2004-2007 UNCW. All rights reserved.

Permission is hereby granted, without written agreement and without license or
royalty fees, to use, copy, modify, and distribute this software and its
documentation for any purpose, provided that the above copyright notice and the
following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL UNCW BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT,
SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF
THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF UNCW HAS BEEN ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.

UNCW SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS"
BASIS, AND UNCW HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT,
UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
*/
```
package org.jxpl.primitives.service;
import org.jxpl.*;
import org.jxpl.bindings.*;
import org.jxpl.exception.*;
import java.util.List;

/**
*   Maintains Mapping of Parameter and variable
*/
public class Param{
   private JxplElement typeDesc;//Describes general format of the variable
   private JxplSymbol variable;//variable for substitution

   public Param(JxplElement typeDesc, JxplSymbol variable){
      this.typeDesc=typeDesc;
      this.variable=variable;
   }

   public void setTypeDesc(JxplElement typeDesc, JxplSymbol variable){
      this.typeDesc=typeDesc;
   }

   public void setVariable(JxplSymbol variable){
      this.variable=variable;
   }

   public JxplElement getTypeDesc(){
      return typeDesc;
   }

   public JxplSymbol getVariable(){
      return variable;
   }

   public static Param buildParam(JxplElement param, Processor environment)throws JxplException{
         Param operationParam=null;

         //evaluate param to convert symbols or simply build the list
         param=environment.evaluate(param);
```

```
        if(param instanceof JxplList){//then we assume that we have a construct defining both type and symbol
           //recorder.debug("Parameter in: "+param);
           List<JxplElement> def=((JxplList)param).getElements();
           if(def.size()>1){
              JxplElement typeDesc=environment.evaluate(def.get(0));
              JxplSymbol symbol=(JxplSymbol)environment.evaluate(def.get(1));
              operationParam=new Param(typeDesc,symbol);

           }else{//we got a list but it only contained one element.  Would be cleaner implementation to reject this
                //...but in the interest of handling any user input this will be done instead
              JxplSymbol symbol=(JxplSymbol)environment.evaluate(def.get(0));
              operationParam=new Param(null, symbol);

           }
        }else if(param instanceof JxplSymbol){//no typeDescription was provided so we assume anyType
          //recorder.debug("Param was a symbol");
          operationParam=new Param(null, (JxplSymbol)param);

        }else throw new JxplMalformedException("Improper format for parameters");
        return operationParam;
    }

}

/*
Copyright © 2004-2007 UNCW. All rights reserved.

Permission is hereby granted, without written agreement and without license or
royalty fees, to use, copy, modify, and distribute this software and its
documentation for any purpose, provided that the above copyright notice and the
following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL UNCW BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT,
SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF
THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF UNCW HAS BEEN ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.

UNCW SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS"
BASIS, AND UNCW HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT,
UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
*/
package org.jxpl.primitives.service.soap;

import org.w3c.dom.*;

import org.jxpl.*;
import org.jxpl.bindings.*;
import org.jxpl.exception.*;

import java.util.*;
import java.util.jar.*;
import java.math.*;

import javax.xml.rpc.handler.Handler;
import javax.xml.rpc.handler.GenericHandler;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.HandlerRegistry;
```

```
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.SOAPMessage;
import javax.xml.rpc.JAXRPCException;
import javax.xml.rpc.ServiceException;
import javax.xml.namespace.QName;

import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import javax.xml.parsers.*;
import javax.xml.soap.*;
import org.apache.commons.logging.*;

//jxpl-ws
import org.jxpl.primitives.service.*;
import org.jxpl.primitives.service.jxpl2Interface.*;

//Axis2
import org.apache.axis2.description.*;
import org.apache.axis2.AxisFault;


/**
 *  This primitive is designed to produce and package web services at design time.
 *  at runtime it is used to evaluate the proper logic.  This switching of mode is determined
 *  by a flag set in the properties of the primitive.
 *
 *  @author Eric Harris
 *  @version 0.1
 */
public class SOAPService implements org.jxpl.primitives.service.Service{

    private Processor environment;

    private String REPOSITORY=System.getProperty("java.io.tmpdir")+File.separator+"repository";

    //file in which to write the archived war/aar file
    private File pathToWar=null;

    private String namespace="http://jxpl.org/jxpl/services/";
    private String serviceName;
    private List<SOAPOperation> operations;
    private JxplElement script;
    private String path;

    private static Log recorder=LogFactory.getLog(SOAPService.class.getName());

    private boolean packageForDeployment=false;
    //true - Package as .aar file for deployment in another container
    //false - Startup local instance of the Axis2 server and load script

    public void setProcessor(Processor env){
        environment = env;
    }
```

```
/**
*  Produce the script for creating and binding the service
*/
public JxplElement evaluate(JxplElement input) throws JxplException {

   List<JxplElement> list = ((JxplList)input).getElements();
   JxplPrimitive prim = (JxplPrimitive)list.get(0);
   boolean generateService=false;

   List propList = prim.getProperties();

   Hashtable props = PropertyHelper.hash(propList, environment);



   if (props.containsKey("name")){
      //recorder.debug("Setting name for service");
      setServiceName((String)props.get("name"));
      path=REPOSITORY+File.separator+getServiceName()+File.separator;
   }

   if (props.containsKey("namepace")){
      setNamespace((String)props.get("namespace"));
   }

   if(props.containsKey("package")){
      Object pack=props.get("package");

      if(pack instanceof String){
         if(((String)pack).equalsIgnoreCase("true"))//if the property value is true
            packageForDeployment=true;//then set the value true
      }else if(pack instanceof JxplSymbol){
         if(((JxplSymbol)pack).getQName().toString().equalsIgnoreCase("true"))
            packageForDeployment=true;
      }
   }

   if(props.containsKey("generate")){
      Object generate=props.get("generate");

      if(generate instanceof String){
         if(((String)generate).equalsIgnoreCase("true"))//if the property value is true
            generateService=true;//then set the value true
      }else if(generate instanceof JxplSymbol){
         if(((JxplSymbol)generate).getQName().toString().equalsIgnoreCase("true"))
            generateService=true;
      }

      if(props.containsKey("outputFile")){
         String loc=(String)props.get("outputFile");
         pathToWar=new File(loc);
      }
   }


   List jxplOperations=((JxplList)list.get(1)).getElements();

   if(jxplOperations.size()<1)//operations must not have been defined
```

```
        throw new JxplMalformedException("Missing Operation descriptions");

    operations=new LinkedList();//initialize the datastructure for tracking operations
    //parse through and load all operations
    for(int i=1; i<jxplOperations.size(); i++){
        addOperation(parseOperation((JxplElement)jxplOperations.get(i)));
    }

    recorder.info("Service has been parsed");

    //output to be sent to the processor
    JxplElement output;

    if(generateService){
        //System.out.println("Building Service");
        setScript(input);
        output= generateSOAPService() ? Processor.TRUE : Processor.FALSE;
    }
    else{
        output=evaluateCall(((JxplList)list.get(2)).getElements());
    }
    return output;
}

/**
*   This method facilitates the invocation of the function if the service is already generated
*   @param params   set of inputs to be passed into the service including the operation name
*/
private JxplElement evaluateCall(List<JxplElement> params)throws JxplException{
    //System.out.println("Evaluating Call");
    JxplElement elem=null;
    String methodName=null;
    SOAPOperation myOperation=null;

    //parse the name of the operation we wish to invoke
    elem=environment.evaluate(params.get(0));
    if(elem instanceof JxplString)
        methodName=((JxplString)elem).getValue();
    else throw new JxplMalformedException("No method defined to invoke");

    //System.out.println("method: "+methodName);

    //find the function to be invoked
    for(int i=0; i<operations.size(); i++){
        SOAPOperation temp=operations.get(i);
        if(methodName.equalsIgnoreCase(temp.getName())){
            myOperation=temp;
            break;
        }
    }

    //System.out.println("found operation ");

    if(myOperation==null)
        throw new JxplException("Cannot find operation");


    /*  Map the method to a DEFUN caching the function for later use */
    if(environment.getVariable(methodName)==null){
```

```
            myOperation.cacheOperation();
            //System.out.println("Cached");
        }

        /* Generate a new list with the proper function call and evaluate */
        JxplList completeCall=new JxplList();
        completeCall.addElement(new JxplPrimitive(methodName));
        for(int i=1; i<params.size(); i++)
            completeCall.addElement(params.get(i));

        return environment.evaluate(completeCall);
    }

    /**
    *   Parse and create the description information required for building the wrapper
    *   @param elem Element containing the description information
    */
    protected SOAPOperation parseOperation(JxplElement elem) throws JxplException{
        //if a symbol is substituted for the script interpret it here
        SOAPOperation operation=null;

        if(elem instanceof JxplSymbol){
            elem=environment.evaluate(elem);
        }

        if(elem instanceof JxplList){
            JxplElement head=((JxplList)elem).first();
            recorder.debug("First Element in the list was "+head);
            if(head instanceof JxplPrimitive){
                JxplPrimitive prim=(JxplPrimitive)head;
                if(prim.getName().equalsIgnoreCase(".service.soap.SOAPOperation")){
                    operation=SOAPOperation.buildOperation(elem, environment);
                    recorder.debug("Operation Parsed "+operation);
                }else throw new JxplMalformedException("Expected SOAPOperation but got "+prim.getName());
            }else parseOperation(head);
        }else throw new JxplMalformedException("List was expected");//in event that something other than a primitive is
placed here

        return operation;
    }

    /**
    *  Adds an operation to the service
    *  @param Operation to be added to service
    */
    public void addOperation(Operation oper){
        if(oper instanceof SOAPOperation)
            operations.add((SOAPOperation)oper);
    }

    /**
    *   Set the service name for this service
    *   @param servicename Name of the service
    */
    public void setServiceName(String serviceName){
        this.serviceName=serviceName;
    }

    /**
```

```
*   Get the serviceName
*/
public String getServiceName(){
   return serviceName;
}

/**
*   Set the namespace for this service instance
*   @param namespace set the namespace for the respective service
*/
public void setNamespace(String namespace){
   this.namespace=namespace;
}


/**
*   Retuns the fully qualified servicename with namespace included.
*   @return QName
*/
public QName getFullServiceName(){
   return new QName(namespace, serviceName);
}

/**
*   Get the namespace of the service
*/
public String getNamespace(){
   return namespace;
}

/**
*   Return a list of operations attached to this service
*/
public List<SOAPOperation> getOperations(){
   return operations;
}

public void setScript(JxplElement script){
   try{
      this.script=convertScript(script);
   }
   catch(JxplException e){
   }
}

public JxplElement getScript(){
   return script;
}

/**
*   Convert the primitive such that it will evaluate a method call upon evaluation
*   instead of generating a service.  This is accomplished by removing the generate flag.
*   @param script   Script to be converted
*/
private JxplElement convertScript(JxplElement script) throws JxplException{
   List<JxplElement> list = ((JxplList)script).getElements();
   JxplPrimitive prim=((JxplPrimitive)list.get(0));
   Vector<JxplElement> props=prim.getProperties();
```

```
    for(int i=0; i<props.size();i++){
        JxplProperty jxplProp=(JxplProperty)environment.evaluate(props.get(i));
        if(jxplProp.getName().equalsIgnoreCase("generate")){
            props.remove(i);
            break;
        }
    }

    prim.setProperties(props);

    recorder.debug("Transformed: \r\n"+script);
    return script;
 }

/**
*   Will generate service from the JxplDescription
*/
protected boolean generateSOAPService() throws JxplException{
    try{
        new File(path+File.separator+"jxplService").mkdirs();//build directory structure
        new File(path+"META-INF").mkdir();
//        new File(REPOSITORY).deleteOnExit();//ensure this file is removed following execution
                                    //this method doesn't seem to work as advertised...

        generateWrapper(path+File.separator+"jxplService");//generate Wrapper Class for building the wsdl
        recorder.debug("Proxy Class Created");

        generateConfigFile(path);
        recorder.debug("Service Config File Created");

        if(packageForDeployment){
            generateAAR(path);
            cleanup();//delete file upon completion
        }
        else
            startServer();//provide parameters
    }catch(Exception e){
        recorder.error(e);
        throw new JxplException("Error generating soap service..."+e.getMessage());
    }
    return true;
}

/**
*   Include any neccessary code to cleanup temporary files goes here
*/
private void cleanup(){
    new File(path).delete();
}


/**
*   Generate service.xml document for use in configuring the service
*/
protected String generateConfigFile(String path) throws Exception{
    String config="<service name=\""+getServiceName()+"\" scope=\"application\">\r\n";
    config+="<description> Published Jxpl Workflow </description>\r\n";
    config+="<messageReceivers>\r\n";
    config+="<messageReceiver mep=\"http://www.w3.org/2004/08/wsdl/in-only\""+
```

```
                    " class=\"org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver\"/>\r\n";
        config+="<messageReceiver mep=\"http://www.w3.org/2004/08/wsdl/in-out\""+
                    " class=\"org.apache.axis2.rpc.receivers.RPCMessageReceiver\"/>\r\n";
        config+="</messageReceivers>\r\n";
        config+="<parameter name=\"ServiceClass\" locked=\"false\">jxplService."+getServiceName()+"</parameter>\r\n";
        config+="</service>";
        PrintWriter configFile=new PrintWriter(new File(path+"META-INF/services.xml"));
        configFile.print(config);
        configFile.close();
        return config;
    }

    /**
    *  Create a class that will generate a proxy class to map rpc calls to wsdl via Axis2's wsdl generator
    */
    protected boolean generateWrapper(String path) throws Exception{

        File source=new File(path+File.separator+getServiceName()+".java");
        System.out.println("Files are "+source);
        //prevent conflicts from prior version
        if(source.exists())
            source.delete();

        File classFile=new File(path+File.separator+getServiceName()+".class");
        if(classFile.exists())
            classFile.delete();

        String proxyCode=JxplInterface.generateInterface(this);

        PrintWriter os=new PrintWriter(source);
        os.print(proxyCode);
        os.close();

        PrintWriter byteout=new PrintWriter(classFile);
        String[] args=new String[3];
        args[0]="-cp";
        args[1]=System.getProperty("java.class.path");//forward the current classpath to the compiler
        args[2]=source.getAbsolutePath();

        // compile the proxy class...binary required for generation of WSDLs
        if(com.sun.tools.javac.Main.compile(args, byteout)>0){
            recorder.error("Unable to compile wrapper class, check parameters to ensure that the parameter names are non-
numeric");
            throw new JxplMalformedException("Unable to compile wrapper class, check paramters to ensure that the parameter
names are non-numeric");
        }

        byteout.close();
        //source.delete();  //eh....we can keep the source around for now
        recorder.debug("Proxy Class Compiled");
        return true;
    }


    /**
    *   Launch Stand Alone Axis2 Server
    *   are there any better ways to launch this...
    */
    private void startServer()throws Exception{
```

```
    recorder.info("Starting Server");
    String[] args={"-repo", REPOSITORY, "-conf", "configs/axis2.xml"};
    //org.apache.axis2.transport.SimpleAxis2Server.main(args);
}

/**
*   Package everything in a jar file that can be used later for deployment
*   within any J2EE - Axis2 enabled container
*   @param directory directory containing all package contents
*   @param aarFile  File in which to generate the package AAR file
*/
protected boolean generateAAR(String directory, File aarFile) throws IOException{
    //File aarFile=new File(pathToWar+getServiceName()+".aar");
    FileOutputStream fout=new FileOutputStream(aarFile);
    FileInputStream fin;
    JarOutputStream jarOut=new JarOutputStream(new BufferedOutputStream(fout));

    /* Write the directory */
    jarDirectory(jarOut, "", directory);
    jarOut.close();

    return true;
}

/**
*   Package everything in a jar file that can be used later for deployment
*   within any J2EE - Axis2 enabled container
*   @param directory directory containing all package contents
*/
protected boolean generateAAR(String directory) throws IOException{
    if(pathToWar!=null)
        generateAAR(directory, pathToWar);
    else{
        generateAAR(directory, new File(getServiceName()+".aar"));
    }

    return true;
}

/**
*   Recursively write files from the given directory structure to a jar archive file
*   @param jarOut   JarOutputStream provided for the jar file
*   @param pathInjar    How the directory structure will appear in the jar file
*   @param actualPath   The actual path of the directory on the file system.
*/
protected void jarDirectory(JarOutputStream jarOut, String pathInJar, String actualPath) throws IOException{
    FileInputStream fin;
    byte[] buffer=new byte[4096];//allow to read up to 4 kilobytes at a time
    String[] files=new File(actualPath).list();

    if(pathInJar.length()>0){
        pathInJar+="/";
        jarOut.putNextEntry(new JarEntry(pathInJar));
        jarOut.closeEntry();
    }

    for(int i=0; i<files.length; i++){
        if(new File(actualPath+"/"+files[i]).isDirectory()){
            jarDirectory(jarOut, pathInJar+files[i], actualPath+"/"+files[i]);
```

```
            }else{
                fin=new FileInputStream(actualPath+"/"+files[i]);
                jarOut.putNextEntry(new JarEntry(pathInJar+files[i]));
                //read the file in
                int length;
                while((length=fin.read(buffer))>0)
                    jarOut.write(buffer,0,length);

                jarOut.closeEntry();
                fin.close();
            }
        }
    }
}
```

/*
Copyright © 2004-2007 UNCW. All rights reserved.

Permission is hereby granted, without written agreement and without license or
royalty fees, to use, copy, modify, and distribute this software and its
documentation for any purpose, provided that the above copyright notice and the
following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL UNCW BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT,
SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF
THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF UNCW HAS BEEN ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.

UNCW SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS"
BASIS, AND UNCW HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT,
UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
*/
```
package org.jxpl.primitives.service.jxpl2Interface;

import org.jxpl.*;
import org.jxpl.bindings.*;
import org.jxpl.primitives.service.*;
import org.jxpl.exception.*;
import java.util.*;


/**
*   Generate the wrapper required to produce the needing functionality for the function.
*   This class will package the script AND then use
*/
public class JxplInterface{

   private String name, qualifiedName;
   private List<JxplMethod> methods=new LinkedList();
   private static JxplTypeTable typeConverter=new JxplTypeTable();
   private JxplElement script;

   public void setQualifiedName(String name){
      qualifiedName=name;
   }

   private void addMethod(JxplMethod elem){

      methods.add(elem);
   }

   /**
   *   Write the source code required for wrapper class.
   */
   public String writeSource(JxplElement script){
      String interfaceString="package jxplService;\r\n"+
                        "import org.jxpl.primitives.service.jxpl2Interface.*;\r\n"+
                        "import org.jxpl.*;\r\n"+
                        "import org.jxpl.bindings.*;\r\n"+
                        "import org.jxpl.primitives.*;\r\n"+
                        "import org.jxpl.exception.*;\r\n\r\n";
```

```
        interfaceString+="public class "+name+"{";
        interfaceString+="\t\tprivate Processor environment=new Processor();\r\n\r\n";

        String xml=script.toString().replace("\r","").replace("\n","").replace("\"", "\\\"");
        interfaceString+="\t\tprivate String xml=\""+xml+"\";\r\n";

        Iterator<JxplMethod> itr=methods.iterator();
        while(itr.hasNext()){
            interfaceString+="\r\n\t"+fillMethod(script, itr.next());
        }
        interfaceString+="\r\n}";

        return interfaceString;
    }

    /**
    *   Take the method header and construct the functional implementation
    */
    private String fillMethod(JxplElement script, JxplMethod m){
        String impl="";
        impl+=m.toString()+"{\n\n";
        impl+="\t\tObject[] params={";

        //add parameters as an array
        List<JxplParam> params=m.getParams();
        for(int i=0; i<params.size(); i++){
            impl+=params.get(i).getName();
            if(i<params.size()-1)
                impl+=",";
        }
        impl+="};\r\n\n";
        impl+="\t\ttry{\r\n\n";
        impl+="\t\t\treturn ("+m.getReturnType()+") WrapperUtil.evaluate(environment, xml, \""+m.getName()+"\", params,
\""+m.getReturnType()+"\");\r\n";
        impl+="\t\t}catch (JxplException e){return null;}\r\n\n";
        impl+="\t}";
        return impl;
    }

    public void setName(String name){
        this.name=name;
    }


    public static String generateInterface(Service s){
        //parse through jxpl element

        JxplInterface intfce=new JxplInterface();
        intfce.setName(s.getServiceName());
        intfce.setScript(s.getScript());
        List<Operation> l=s.getOperations();

        for(int i=0; i<l.size(); i++){

            Operation oper=l.get(i);

            //change this later to reflect correct typing
            JxplMethod meth=new JxplMethod(oper.getName(),
                        typeConverter.getJxpl2Java(oper.getReturnType()));
```

```
            List<Param> prms=oper.getParameters();

            for(int j=0; j<prms.size(); j++){
               Param p=prms.get(j);
               JxplParam param=new JxplParam(p.getVariable().getQName().toString(),
                                 typeConverter.getJxpl2Java(p.getTypeDesc()));
               meth.addParam(param);
            }
            intfce.addMethod(meth);
         }

         return intfce.writeSource(s.getScript());
      }

      public void setScript(JxplElement script){
         this.script=script;
      }

}

class JxplMethod{
   private String name;
   private String returnType="void";
   private List<JxplParam> params=new LinkedList();


   public JxplMethod(String name, String returnType){
      this.name=name;
      this.returnType=returnType;
   }

   public void addParam(JxplParam in){
       params.add(in);
   }

   public String toString(){
      String methodString="public "+returnType+" "+name+"(";
      Iterator itr=params.iterator();
      while(itr.hasNext()){
         methodString+=itr.next().toString();
         if(itr.hasNext())
            methodString+=",";
      }
      methodString+=")";
      return methodString;
   }

   public String getName(){
      return name;
   }

   public String getReturnType(){
      return returnType;
   }

   public List<JxplParam> getParams(){
      return params;
   }
```

```
}

class JxplParam{
   String type;
   String name;

   public JxplParam(String name, String type){
      this.name=name;
      this.type=type;
   }

   public JxplParam(String name){
      this(name, "Object");
   }

   public void setType(String type){
      this.type=type;
   }

   public void setName(String name){
      this.name=name;
   }

   public String getName(){
      return name;
   }

   public String getType(){
      return type;
   }

   public String toString(){
      return type+" "+name;
   }


   /*
   public static JxplParam buildParam(JxplElement elem){
      JxplParam param=new JxplParam();
      List list = ((JxplList)elem).getElements();
      JxplPrimitive prim = (JxplPrimitive)list.get(0);
      List propList = prim.getProperties();
      Hashtable props = PropertyHelper.hash(propList, environment);

      if(list.size()>2){//we have both name and type
         JxplParam param=new JxplParam(list.get(2));
         param.setType(Jxpl2JavaTypeMap.getType((JxplElement)list.get(1)));
      }
      else
         param=new JxplParam(list.get(1));
   }
   */
}
```

/*
Copyright © 2004-2007 UNCW. All rights reserved.

Permission is hereby granted, without written agreement and without license or
royalty fees, to use, copy, modify, and distribute this software and its
documentation for any purpose, provided that the above copyright notice and the
following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL UNCW BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT,
SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF
THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF UNCW HAS BEEN ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.

UNCW SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS"
BASIS, AND UNCW HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT,
UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
*/
```
package org.jxpl.primitives.service.jxpl2Interface;

import org.jxpl.Processor;
import org.jxpl.bindings.*;
import org.jxpl.primitives.*;
import org.jxpl.exception.*;
import org.w3c.dom.Document;
import javax.xml.parsers.*;
import org.apache.axiom.om.impl.builder.*;
import java.io.*;

/**
*   Utility class to aide in the mapping and evaluation in the JxplProcessor
*   @author Eric Harris
*/
public class WrapperUtil{

   /**
   *  This method makes the connection between the auto generated wrapper class and the logic that drives it
   *  @param environment   JxplProcessor required for evaluation
   *  @param methodName    Name of the function being invoked
   *  @param parameters    Array of parameters for the method invocation
   */
   public static Object evaluate(Processor environment, String scriptIn, String methodName, Object[] parameter, String
returnType) throws JxplException{

      JxplList l=(JxplList)getElementFromString(scriptIn);
      JxplList call=new JxplList();
      call.addElement(new JxplString(methodName));
      System.out.println(methodName);
      System.out.println(parameter.length);
      System.out.println("Wrapper Input is : "+parameter[0]);
      for(int i=0; i<parameter.length; i++){//not very elegant especially if more types need to be added!!!
         call.addElement(JxplTypeTable.convertJava2Jxpl(parameter[i]));
      }
      l.addElement(call);


      JxplElement elem=environment.evaluate(l);
```

```
      System.out.println("Output from Wrapper is "+elem.toString());
      Object out;
      if(returnType.equalsIgnoreCase("java.lang.Boolean"))//special case
         out=JxplTypeTable.convertJxpl2Java(elem, true);

      else out=JxplTypeTable.convertJxpl2Java(elem);

      return out;
   }

   /**
   *   Extract a JxplElement from a String
   */
   private static JxplElement getElementFromString(String s) throws JxplException
   {
      JxplList output=null;
      byte [] ary=s.getBytes();
      ByteArrayInputStream bytes=new ByteArrayInputStream(ary);

      try
      {
         DocumentBuilderFactory dfac=DocumentBuilderFactory.newInstance();
         DocumentBuilder dBuilder=dfac.newDocumentBuilder();
         Document input=dBuilder.parse(bytes);
         output=new JxplList(input.getDocumentElement());
      }
      catch(Exception se)
      {
         throw new JxplException(se.getMessage());
      }
      return output;
   }
}
```

/*
Copyright © 2004-2007 UNCW. All rights reserved.

Permission is hereby granted, without written agreement and without license or
royalty fees, to use, copy, modify, and distribute this software and its
documentation for any purpose, provided that the above copyright notice and the
following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL UNCW BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT,
SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF
THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF UNCW HAS BEEN ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.

UNCW SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS"
BASIS, AND UNCW HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT,
UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
*/
```
package org.jxpl.primitives.service.jxpl2Interface;

import org.jxpl.bindings.*;
import org.jxpl.exception.*;
import org.jxpl.*;
import java.util.*;
import org.w3c.dom.Element;
import org.apache.axiom.om.*;
import org.apache.axiom.om.impl.dom.*;
import org.apache.axiom.om.impl.builder.*;
import org.apache.axiom.om.impl.dom.factory.OMDOMFactory;
import javax.xml.stream.*;
import java.io.*;


/**
* Maintain a mapping between Jxpl and Java types.
* This mapping helps maintain a relationship between the basic JxplBindings and their simplyType
* counterparts.  The goal of this class is to 1) enable safe mapping in the auto generated wsdl
* and 2) to provide a safe construct in which datatypes may be wrapped and converted from their
* basic java counterparts (as the WSDL generator goes from Java -> WSDL). It can be argued that
* some of the mapping isn't neccessary.  However, I didn't want the basic types to all map as
* complexTypes when simple types could suffice.
*
* @author Eric Harris
*/
public class JxplTypeTable{

   private HashMap<String,String> jxplTypetoJavaType, javaTypetoJxplType;

   public JxplTypeTable(){

      jxplTypetoJavaType=new HashMap();
      javaTypetoJxplType=new HashMap();
      populateJxplTypes();
   }

   /**
   *  Populate the tables with the type match...this is used by classes such as JxplInterface to ensure that
   *  the appropiate casting is performed.
```

```
   */
   private void populateJxplTypes(){

      //jxpl -> java
      jxplTypetoJavaType.put("org.jxpl.bindings.JxplString", "java.lang.String");
      jxplTypetoJavaType.put("org.jxpl.bindings.JxplInteger", "java.lang.Integer");
      jxplTypetoJavaType.put("org.jxpl.bindings.JxplDecimal", "java.lang.Double");
      jxplTypetoJavaType.put("org.jxpl.bindings.JxplXml", "org.apache.axis2.om.OMElement");  //this isn't neccesary but
leaving it in for reference

      //java -> jxpl
      javaTypetoJxplType.put("java.lang.String", "org.jxpl.bindings.JxplString");
      javaTypetoJxplType.put("java.lang.Integer", "org.jxpl.bindings.JxplInteger");
      javaTypetoJxplType.put("java.lang.Double", "org.jxpl.bindings.JxplDecimal");
      javaTypetoJxplType.put("org.apache.axis2.om.OMElement", "org.jxpl.bindings.JxplXml");   //this isn't neccesary but
leaving it in for reference
   }


   /**
   *   Discover the corresponding java type
   *   if not known then assume it as an object
   *   @param elem object of jxpl type
   */
   public String getJxpl2Java(JxplElement elem){
      String out=getJxpl2Java(elem.getClass().getName());
      return out!=null ? out : "Object";
   }

   /**
   *   Discover the corresponding java type
   *   If the element is not known than assume a generic object
   *   @param string name of java type
   */
   public String getJxpl2Java(String string){
      String out=jxplTypetoJavaType.get(string);
      return out!=null ? out : "Object";
   }

   /**
   *   Discover the corresponding jxpl type. If one cannot be found substitute a jxplList
   *   @param string name of jxpl type
   */
   public String getJava2Jxpl(String string){
      String out=javaTypetoJxplType.get(string);
      return out!=null ? out : "org.jxpl.bindings.JxplList";
   }

   /**
   *   Discover the corresponding jxpl type
   *   @param obj  Object of java type
   */
   public String getJava2Jxpl(Object obj){
      if(obj instanceof JxplElement){//if it is a jxpl type then leave it alone
         return obj.getClass().getName();
      }

      String out=javaTypetoJxplType.get(obj.getClass().getName());
      return out!=null ? out : "org.jxpl.bindings.JxplList";//if it is not
```

```
}

/**
*   If we don't know what the object is...set it as a blank JxplList
*   @param obj  Object to convert
*/
public static JxplElement convertJava2Jxpl(Object obj) throws JxplMalformedException{
   System.out.println("Inputted "+obj);
   JxplElement out=new JxplList();

   if(obj instanceof Integer)
      out=convertJava2Jxpl((Integer)obj);
   else if(obj instanceof Long)
      out=convertJava2Jxpl((Long)obj);
   else if(obj instanceof Double)
      out=convertJava2Jxpl((Double)obj);
   else if(obj instanceof String)
      out=convertJava2Jxpl((String)obj);
   else if(obj instanceof Boolean)
      out= (Boolean)obj ? org.jxpl.Processor.TRUE : org.jxpl.Processor.FALSE;
   else if(obj instanceof Element)
      out=convertJava2Jxpl((Element)obj);
   return out;
}

public static JxplInteger convertJava2Jxpl(Integer i){
   return new JxplInteger(i);
}

public static JxplInteger convertJava2Jxpl(Long i){
   return new JxplInteger(i);
}

public static JxplString convertJava2Jxpl(String s){
   return new JxplString(s);
}

public static JxplDecimal convertJava2Jxpl(Double d){
   return new JxplDecimal(d);
}

public static JxplXml convertJava2Jxpl(Element e){
   //System.out.println("Got a JxplXml");
   return new JxplXml(e);
}

public static Object convertJxpl2Java(JxplElement obj)throws JxplMalformedException{
   return convertJxpl2Java(obj,false);
}

/**
*  Convert the Jxpl to its corresponding Java Type
*/
public static Object convertJxpl2Java(JxplElement obj, boolean expectBool)throws JxplMalformedException{

   Object out=obj;//default object
   if(obj instanceof JxplInteger)
      out=convertJxpl2Java((JxplInteger)obj);
   else if(obj instanceof JxplString)
```

```
            out=convertJxpl2Java((JxplString)obj);
        else if(obj instanceof JxplDecimal)
            out=convertJxpl2Java((JxplDecimal)obj);
        else if(obj instanceof JxplXml)
            out=convertJxpl2Java((JxplXml)obj);
        else if(expectBool){//when expecting a boolean output JxplSymbol w\ value "true" and an empty jxpl list have different
meanings
            if(obj instanceof JxplSymbol){
                if(((JxplSymbol)obj).getQName().getLocalPart().equalsIgnoreCase("true"))
                    out=new Boolean(true);
            }
            else if(obj instanceof JxplList)
                out=new Boolean(false);
        }

        else try{
            out=toOM(XmlUtil.getDocument(obj).getDocumentElement());
        }
        catch(Exception e){
            throw new JxplMalformedException(e);
        }

        return out;
    }

    public static Integer convertJxpl2Java(JxplInteger i){
        return new Integer(i.getValue().intValue());
    }

    public static Double convertJxpl2Java(JxplDecimal d){
        return new Double(d.getValue().doubleValue());
    }

    public static String convertJxpl2Java(JxplString s){
        return s.getValue();
    }

    public static OMElement convertJxpl2Java(JxplXml x)throws JxplMalformedException{
        return toOM(x.getElement());
    }

    /**
    *  Utility function mapping the older DOM element org.w3c.Element
    *  to the DOM element used in axis2 OMElement
    *  @param Element   DOM Element to be converted
    */
    private static OMElement toOM(Element input)throws JxplMalformedException{
        try{
            String xml=XmlUtil.dumpElement(input);
            javax.xml.stream.XMLStreamReader reader=XMLInputFactory.newInstance().createXMLStreamReader(new
StringReader(xml));
            StAXOMBuilder ombuild=new StAXOMBuilder(reader);
            return ombuild.getDocumentElement();
        }
        catch(Exception e){
            throw new JxplMalformedException(e);
        }

    }
```

```
}
```

/*
Copyright © 2004-2007 UNCW. All rights reserved.

Permission is hereby granted, without written agreement and without license or
royalty fees, to use, copy, modify, and distribute this software and its
documentation for any purpose, provided that the above copyright notice and the
following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL UNCW BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT,
SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF
THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF UNCW HAS BEEN ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.

UNCW SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS"
BASIS, AND UNCW HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT,
UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
*/
```
package org.jxpl.primitives.ws;

import org.jxpl.*;
import org.jxpl.bindings.*;
import org.jxpl.exception.*;
import java.util.*;
import java.math.*;
import java.net.*;
import javax.xml.namespace.QName;
import javax.xml.stream.*;
import org.apache.axis2.util.XMLUtils;

//for understanding the wsdl
import javax.wsdl.factory.WSDLFactory;
import javax.wsdl.xml.WSDLReader;
import javax.wsdl.*;

import java.io.*;
import org.apache.axiom.om.impl.dom.*;
import org.apache.axiom.om.impl.builder.*;
import org.apache.axiom.om.impl.dom.factory.OMDOMFactory;
import org.apache.axiom.om.*;
import org.apache.axis2.AxisFault;
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.apache.axis2.rpc.client.RPCServiceClient;
import org.apache.axis2.saaj.util.SAAJUtil;
import org.apache.commons.logging.*;

import org.w3c.dom.*;


/**
 * This implementation of a generic Web Service Client is based on
 * models presented by the Axis2 API.  This implementation DOES NOT support reading complex types.
 * at present part type namespaces are assumed to be the same.
 *
 * @author Eric Harris
 * @version 1.0
```

```
 */
public class WSClient implements Primitive{
  private Processor environment;
  //private ServiceClient serviceClient;
  private String namespace;
  private Definition wsdlDefinition;
  private boolean security=false;
  private static Log recorder=LogFactory.getLog(WSClient.class);

  public void setProcessor(Processor env){environment = env;}


  public WSClient(){

  }

  public WSClient(String namespace){
    setNamespace(namespace);
  }



  public JxplElement evaluate(JxplElement input) throws JxplException {
     List<JxplElement> list=((JxplList)input).getElements();

     JxplPrimitive prim = (JxplPrimitive)list.get(0);
     List propList = prim.getProperties();
     Hashtable props = PropertyHelper.hash(propList,environment);

     if(props.containsKey("WS-Security")){
       Object sec=props.get("WS-Security");

          if(sec instanceof String){
             if(((String)sec).equalsIgnoreCase("true"))//if the property value is true
                security=true;//then set the value true
          }else if(sec instanceof JxplSymbol){
             if(((JxplSymbol)sec).getQName().toString().equalsIgnoreCase("true"))
                security=true;
          }
     }

     /* Get the values */
     EndpointReference endpoint = new EndpointReference(((JxplString)list.get(2)).getValue());
     URL wsdl;
     try{
       wsdl=new URL(((JxplString)list.get(1)).getValue());//wsdl
     }
     catch(Exception e){
       throw new JxplException("WSDL Cannot be found");
     }

     String operation = ((JxplString)list.get(3)).getValue();//operation name


     Options options = new Options();
     options.setTo(endpoint);

     if(security){//add options for ws security
       recorder.info("ws-security...not yet implementated");
```

```
      //NamedStaxOMBuilder builder = new StAXOMBuilder("");
      //Policy clientPolicy = PolicyEngine.getPolicy(builder.getDocumentElement());
      //options.setProperty(RampartMessageData.KEY_RAMPART_POLICY, clientPolicy);
    }

    Element resp;
    OMElement result;

    try{
      wsdlDefinition=getWsdlDefinition(wsdl);

      setNamespace(wsdlDefinition.getTargetNamespace());
      ServiceClient serviceClient=new ServiceClient();
      serviceClient.setOptions(options);

      OMElement message=generateInput(operation, list.subList(3, list.size()));

      //Finally send the message and get a response
      result = serviceClient.sendReceive(message);
      //System.out.println(result);

      //resp=OMElementToDocument(result);
      //resp=SAAJUtil.toDOM(result.getFirstElement());

    }catch(Exception e){
      throw new JxplException(e);
    }

    try{
      return convertResults(result);
    }
    catch(JxplException ex) {//this should never get thrown but just in cast I will leave it for now
      //Element soapRoot = resp.getDocumentElement();
      try{
         return new JxplXml(SAAJUtil.toDOM(result));
      }catch(Exception e){
         return null;
      }
    }
}

protected void setNamespace(String namespace){
  this.namespace=namespace;
}

protected String getNamespace(){
  return namespace;
}

/**
* For the provided operation read the wsdl to determine what the appropiate namespace for its "parts" (parameters).
* If the "parts" of the operation request are incorrect the service will fail to understand it (and return a fault).
* However, a limitation here is that we assume all "parts" of an operation request have the same namespace.
*
* @param operationName    Operation being referenced
*/
protected String getPartNamespace(String operationName) throws JxplException{

      PortType pt=(PortType)wsdlDefinition.getPortTypes().values().toArray()[0];
```

```
    Operation oper=pt.getOperation(operationName, null,null);
    if(oper==null)
       throw new JxplMalformedException("No Such Operation Exists");
    List<Part> parts=oper.getInput().getMessage().getOrderedParts(null);
    if(parts.size()>0)
       return parts.get(0).getElementName().getNamespaceURI();
    else return getNamespace();
}


/**
* Read in the wsdl for further examination
* @param wsdlPath
*/
protected Definition getWsdlDefinition(URL wsdlPath) throws Exception{
  InputStream in = wsdlPath.openConnection().getInputStream();
  Document doc = XMLUtils.newDocument(in);
  WSDLReader reader = WSDLFactory.newInstance().newWSDLReader();
  reader.setFeature("javax.wsdl.importDocuments", true);
  Definition wsdlDefinition = reader.readWSDL(null, doc);
  return wsdlDefinition;
}

/**
*    Construct the message
*    @param operationName Name of the operation being invoked
*    @param inputs being passed into the operation...should be one for this iteration
*/
protected OMElement generateInput(String operationName, List<JxplElement> inputs) throws JxplException{
    OMFactory fac = OMAbstractFactory.getOMFactory();

    //for now lets assume the namespace of the first part is the same for all (no mixing of types)...
    OMNamespace omNs = fac.createOMNamespace(getPartNamespace(operationName), "jc");
    OMElement method = fac.createOMElement(operationName, omNs);

    for(int i=1; i<inputs.size(); i++){
        JxplElement elem=environment.evaluate(inputs.get(i));

        //Special Case...if the parameters consist only of a blank list then assume no parameters
        if(i==1 && elem instanceof JxplList && ((JxplList)elem).getElements().size()==0){
           break;
        }
        OMElement value = fac.createOMElement("part"+i, omNs);
        value.addChild(createChild(elem, fac, value));
        method.addChild(value);
    }

    return method;
}

/**
*     Create a child to pass into the xml document from the proper jxpl type
*     @param in   JxplElement to be converted and added to the message
*     @param fac  OMFactory used to append new components onto the soap message
*     @param node OMNode under which to add the element
*/
protected OMNode createChild(JxplElement in, OMFactory fac, OMElement node)throws JxplException{
       if(in instanceof JxplXml){
```

```
            return OMUtil.toOM(((JxplXml)in).getElement());
            //throw new JxplMalformedException("Support for complexTypes not implemented");
         }
         else if(in instanceof JxplList)
            try{
               return OMUtil.toOM(XmlUtil.getDocument(in).getDocumentElement());
            }
            catch(Exception e){
               throw new JxplException(e);
            }
         else return fac.createOMText(node, Util.convertFromJxpl(in).toString());
  }

  /**
  *     @deprecated This method was part of an abandoned approach to jxplList handling, but it is being kept around for the
moment
  */
  protected void addFromJxplList(OMElement n, OMFactory fac, JxplList in, String name)throws JxplException{
       List<JxplElement> l=((JxplList)in).getElements();
       for(int j=1; j<l.size();j++){
          OMElement value=fac.createOMElement(name, n.getDefaultNamespace());
          value.addChild(createChild(l.get(j), fac, value));
          n.addChild(value);
       }
  }

  /**
  *     Convert the Response message into a JxplElement
  *     @param result XML received from the Server
  */
  protected JxplElement convertResults(OMElement result) throws JxplException{
    String value=result.getFirstElement().getText();


    //if the result is an XML document
    if((value==null || value=="") && result.getChildElements().hasNext()){
       try{
          return new JxplXml(SAAJUtil.toDOM(result.getFirstElement()));
       }catch(Exception e){
          throw new JxplMalformedException("Problem parsing as JxplXML "+e.getMessage());
       }
    }

    //else try and see if its an integer
    try{
       return new JxplInteger(Integer.parseInt(value));
    }catch(NumberFormatException nfi){
       //not an integer
    }

    //what if it is a decimal value
    try{
       return new JxplDecimal(Double.parseDouble(value));
    }catch(NumberFormatException nfi){
       //not a double
    }

    //maybe boolean...
    if(value.equalsIgnoreCase("true"))
```

```
      return environment.TRUE;
   else if(value.equalsIgnoreCase("false"))
      return environment.FALSE;
   else //it must be a string
      return new JxplString(value);
  }

}

/**
*   Custom class for facilitating the conversion from w3c xml model to Axiom model
*
*/
class OMUtil{

   /**
   *   Convert the W3c Document Element into a AXIOM OMElement<br>
   *   <p>
   *   &lt;<b>Notes:</b>&gt;
   *   <ul>
   *   <li>W3C Document Element is the older Document Object Model used by JXPL</li>
   *   <li>Axiom OMElement is the newer model used by AXIS2</li>
   *   </ul>
   *   </p>
   */
   public static OMElement toOM(Element input)throws JxplMalformedException{
      try{
         String xml=XmlUtil.dumpElement(input);
         XMLStreamReader reader=XMLInputFactory.newInstance().createXMLStreamReader(new StringReader(xml));
         StAXOMBuilder ombuild=new StAXOMBuilder(reader);
         return ombuild.getDocumentElement();
      }
      catch(Exception e){
         throw new JxplMalformedException(e);
      }

   }
}
```