A Generic Framework for Building Services and Scientific Workflows for the Grid

Gopi Kandaswamy, Liang Fang, Yi Huang, Satoshi Shirasuna and Dennis Gannon
(Department of Computer Science, Indiana University, Bloomington, Indiana)

Abstract— **Web Service architectures have gained popularity in recent years because they allow software and services from various organizations to be combined easily to provide integrated and distributed applications. However, most applications developed and used by scientific communities are not Web Service oriented and there is a growing need to integrate them into Grid applications based on Web Service architectures. But are scientific communities willing to rewrite each of their applications as a Web Service? Do they want to write complex software to manage user authentication and authorization? Do they want to write complex interfaces between applications to manage interactions between them? Can they afford to keep up with rapidly changing service-oriented technologies today? What they do want is to be able to do experiments of ever increasing complexity using scientific workflows. In this paper we describe a framework that allows scientists to wrap their existing applications as Web Services without having to write any extra code or modify their applications in any way. The framework also allows scientists, educational users and other users to discover these application services, interact with them and compose scientific workflows from the convenience of a Grid Portal.**

Index Terms— **Grid Services, Grid Workflows, Grid Portals, Grid Security**

I. Introduction

Because of the role played by Grid technologies in large-scale scientific collaborations, Web Service architectures have grown in significance in recent years. Consequently, many scientific communities are feeling a growing need to convert their legacy applications into Web Services. Unfortunately, most of the applications developed and used by scientific communities are command-line applications written in FORTRAN, C and a host of scripting languages. They are fast, efficient and easy to use. However, they are usually platform dependent and are difficult to integrate with applications from other communities. There is no standard way of registering these applications so that they can be discovered by interested clients and end-users. Programmatic access from remote clients is difficult. Many of them lack a graphical user interface, which makes it cumbersome for end-users to interact with them. Also, there is no standard way to describe their input parameters and output results and to monitor their progress as they run for extended periods of time on the Grid. By converting these command-line applications into application services, we can overcome most of the aforesaid limitations. An application service is an application with a Web Service interface to it. The Web Service interface is described in the Web Service Definition Language (WSDL) [1] as a set of endpoints operating on messages containing document-oriented information. We discuss our experiences with a framework we have built that allows scientists to not only wrap their applications as services and deploy them on the Grid, but also securely interact with these services, compose scientific workflows using these services and monitor the status of their workflows on the Grid. The framework has four primary components:

- A Grid Portal, which is a Web Server and a gateway for users to access services, compose workflows and manage data.
- A generic Factory [2] Service that is invoked from the Portal by application providers to wrap applications as services and create new instances of these services on the Grid.
- A workflow composer tool that allows users to compose complex and interesting workflows from application services.
- A Notification Service that allows application services to send messages that are logged by the Portal and monitored by the workflow instance.

Our primary focus in this paper is on the design of the application services and the associated security architecture. We begin by discussing some related work in section II. In section III we show how application services can be created and accessed from a Grid Portal. We discuss the architecture of the application services in section III-C. In section III-D we describe the authentication and authorization mechanisms implemented by these services. We then show how users can compose scientific workflows from these services in section IV.

II. Related Work

Several frameworks have been developed to compose and run scientific workflows on the Grid. However, most of them do not support wrapping an application as a Web Service. SoapLab [3], [4] is a set of Web Services that provides programmatic access to some applications on remote computers. It can create two types of Web Service; Analysis Service and Derived Analysis Service. While the former allows users to send input data as weakly typed name-value pairs, the later has strongly typed methods for sending input data and receiving results. SoapLab uses Apache Axis [5] to create Java implementation classes and deployment descriptors for all Derived Analysis Services. It uses CORBA [6] on the server side for finding, starting, controlling and using applications. Although SoapLab serves to wrap as a Web Service almost any command-line tool, it has a number of limitations. SoapLab does not have a rigorous Notification Service that can accept CORBA events and propagate them to clients. This makes monitoring the status of services and workflows difficult. SoapLab is not Grid enabled and service-level authentication and authorization are not addressed. Also, the basic problem with code generation and deployment is that if the server side logic changes, then the implementation classes have to be re-generated and re-deployed. This can be a time consuming process for large scientific workflows involving hundreds of services.

Gowlab [7] is an application that enables ordinary Web pages to be wrapped as Web Services. It also allows programmatic access to these services. However, these services are difficult to maintain because of the non-standardized and changeable nature of Web pages. Also, most Web pages are non-trivial and require the Gowlab service provider to write Java implementation classes to extract information from them.

Pise [8] is a great tool for generating Web interfaces for Molecular Biology applications. It does not wrap applications as Web Services but merely adds a friendly user interface to them. It supports a variety of interfaces like HTML, Tcl/Tk and X11. It also allows users to compose workflows from them. However, Pise does not handle authentication and authorization issues that arise when users access application services that run under the credentials of the application provider. It does not have a rigorous notification mechanism. Moreover, it is a toolkit specific to Molecular Biology applications and is not very extensible.

The GridLAB [9] project aims to provide application tools and middleware for Grid environments. It uses the Grid Application Toolkit (GAT) [10] which is set of APIs that Grid application programmers can use for uniformly accessing numerous Grid Services and middleware. However, GAT does not address the problem of wrapping existing applications as Web Services.

Kepler [11] is an open-source scientific workflow system. It allows scientists to design and execute scientific workflows on the Grid. It features a generic Web Service Actor which can take the URL of a WSDL and instantiate any operation specified in the WSDL. After instantiation, the Web Service Actor can be used in a scientific workflow as a local component. However, a Web Service Actor merely serves as a client to Web Services. It does not create Web Service wrappers for applications.

There are a number of others frameworks like SeqHound [12], BioMoby [13] and Kegg [14] that are specific to Bio-Informatics and are not extensible.

III. Building Services from applications

Our framework provides the *Generic Service Toolkit* to enable application providers to wrap their applications as secure services. The service and the application run under the identity of the application provider, who authorizes interested users access to her service. If every user of a Grid Portal wanted to run her own instance of the service she wanted to access, there would be too many independent and persistent services for the Grid to handle. From our experience we have realized that it is unrealistic to get a large number of services running independently and persistently without a huge commitment in the form of resources and support infrastructure. However, we feel that it is possible to support a small number of Factory Services running persistently that can dynamically create instances of other transient and possibly stateful services on the Grid. The *Generic Service Toolkit* features the *Generic Factory Service* and the *Generic Service Client*. The *Generic Factory Service* is a stateless and persistent Web Service that can be used for creating and deploying application services on the Grid. The *Generic Service Client* can be used to interact with the Factory and the application services that it creates. Application providers, who want to wrap their applications as a service, need to give the Factory a description of the service. Our Factory uses a schema that we call the *ServiceMap,* for describing a service.

A. The ServiceMap Schema

The ServiceMap schema contains information about the service port-types, including their operations and the input and output parameters. It also contains configuration parameters that are needed to instantiate the service. A ServiceMap document that conforms to the ServiceMap schema has four main elements; *service*, *portType, creationParameter* and *policy*. The *service* element describes the service to be created. It contains the name of the service and a short description of the service. It also contains metadata about the service. The *metadata* element supports any XML schema. A ServiceMap document for a simple data decoder that translates a data file from one format to another is illustrated below.

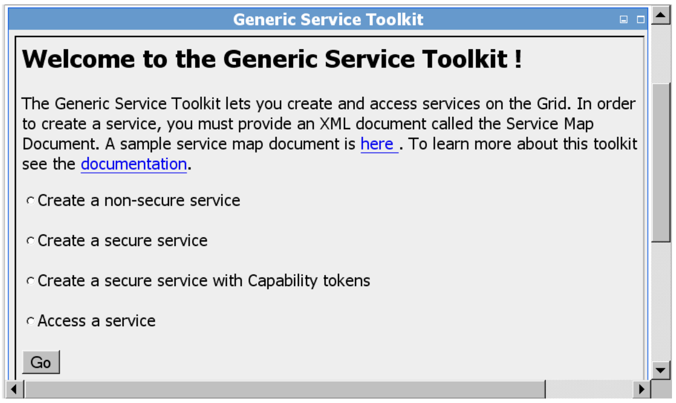
```
<service>
  <serviceName> Decoder </serviceName>
  <serviceDescription>
    Data decoder
  </serviceDescription>
  <metadata> Anything </metadata>
</service>
<portType>
  <portName> DecoderPort </portName>
  <portDescription>
    Port to decode data
  </portDescription>
  <metadata> Anything </metadata>
</portType>
<method>
  <methodName> Run </methodName>
  <methodDescription>
    Runs the decoder
  </methodDescription>
  <metadata> Anything </metadata>
  <script>
    <typeOfScript> ogre </typeOfScript>
    <scriptFile> decoderScript.xml </scriptFile>
  </script>
  <inputParameter>
    <parameterName> topic </parameterName>
    <parameterDescription>
       The topic for notification messages
    </parameterDescription>
    <metadata> Anything </metadata>
  </inputParameter>
</method>
<creationParameter>
  <host> chinkapin.cs.indiana.edu </host>
  <workDir> /tmp/decoder </workDir>
</creationParameter>
```

The *portType* element contains a name, description, metadata and a list of operations (also known as methods). Each operation has a name, a description and metadata. It also contains an application which is either a simple command-line executable like a UNIX command or a more sophisticated script written in OGRE [15], Jython, Python, Perl etc. When an operation is invoked on the service, the application is invoked. The input parameters to the application are specified as input parameters to the operation. Each input parameter has a name, description, metadata and a list of default values. In the current implementation, the input parameter values must be strings. In future we plan to support complex object types. Default values are specified using the *parameterValue* element. Security policy information can be specified using the *policy* element. In the current implementation, the policy specification is quite primitive and allows the application provider to specify a list of users and groups who are authorized to access the service and a lifetime for the policy. We will discuss more about this in section III-D. The *creationParameter* element specifies the physical location where the service will be started. It contains the host and the working directory of the service.

B. Creating and Accessing Services from the Portal

The standard Grid Portals we describe here are based on the JSR-168 portlet container model [16]. When a user authenticates with the Portal, a "context" is created for that session of the user. What the user sees is a set of portlets that each have a user interface and some back-end logic that runs in the Portal Server. The application provider uses the *Generic Factory Service* to wrap an application as a Web Service. To do so, she first uses the Proxy-Manager Portlet to load her Grid proxy certificate into her Portal context. She then uses the *Generic Service Client* (also known as the *Generic Service Portlet* shown in Figure 1) to access the Factory. To create a service, she uploads the ServiceMap document to the portlet which it then transfers to the Factory. After validating the ServiceMap document, the Factory creates and starts the service on the specified host using Globus Resource Allocation Manager (GRAM) [17], [18]. To do so it needs the proxy certificate of the application provider which it obtains from her Portal context. After it is instantiated, the service registers its WSDL with a Registry Service. This allows the service to be discovered by interested clients and end-users that search for it in the Registry using its name or metadata. The Registry returns the service's WSDL that the clients use to access the service. While client programs such as workflows access the service programmatically, end-users rely on the graphical user interface that is generated by the service itself; a concept that we borrowed from WSRP [19].

When a user accesses a service using the *Generic Service Portlet*, the portlet requests the service for a user interface. The service then creates a user interface in the form of a HTML page and sends it back to the portlet which is displayed to the user. A sample user interface is shown in Figure 2. The user interface shows all the operations that the user is allowed to invoke on the service. When the user selects an operation,


Generic Service Toolkit

Welcome to the Generic Service Toolkit !

The Generic Service Toolkit lets you create and access services on the Grid. In order to create a service, you must provide an XML document called the Service Map Document. A sample service map document is here . To learn more about this toolkit see the documentation.

- Create a non-secure service
- Create a secure service
- Create a secure service with Capability tokens
- Access a service

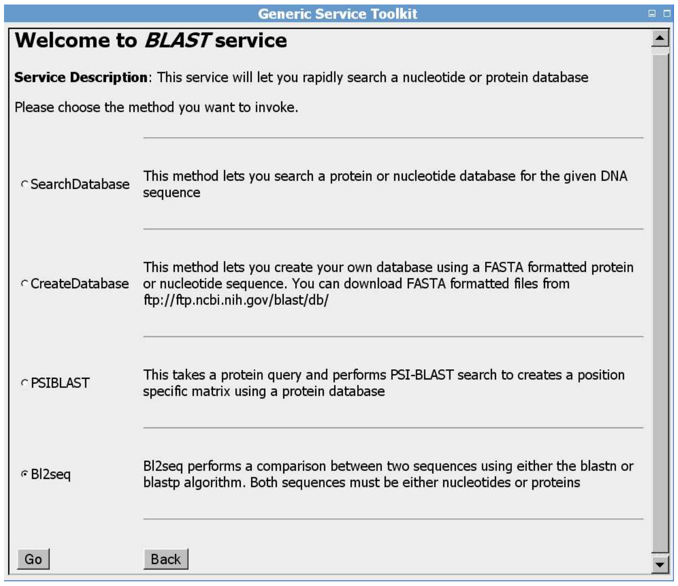
Go

Fig. 1. Generic Service Portlet

the portlet sends another request to the service to obtain the user interface for that operation (Figure 3). It is also a HTML form that the user needs to complete in order to invoke the operation. In the ServiceMap document, the application provider can describe the user interface that she wants her service to create for its users. For example, she can specify the user interface for an input parameter in its metadata. The *formElement* in the metadata of an input parameter tells the application service what “HTML form element” to use to display the default values for that parameter. There are a number of these form elements that are supported; *ListBox, RadioButton*, *CheckBox* and *RemoteFile* to name a few. Some of these have special properties. For example, the *RemoteFile* provides a “browse” button for the user to upload a file to the portlet which will then transfer it to the application service using GridFTP [20]. The ServiceMap document also allows the application provider to specify different user interfaces for different groups of users. This means that while novice users enjoy a simple interface to the service, advanced users can get a richer interface that gives them better control of the applications that these services encompass. We plan to support multiple and complex user interfaces in the next release of our *Generic Service Toolkit*.

After the user specifies all the input parameter values for an operation, the portlet sends a SOAP message to the service to invoke that operation. The service invokes the application and sends notification messages about its status and the status of its application to a Notification Service. Clients and end-users can then listen to these notification messages by subscribing to the topic to which the notification messages are being sent. Figure 4 shows the Notification Viewer that we use from our Portal for viewing notification messages.

The Notification Service we use is called WS-Messenger. It is our implementation of a notification model and is compliant with the WS-Notification [21] and WS-Eventing [22] specifications. It provides a messaging service for Web Services based on the publish/subscribe paradigm. WS-Messenger uses a topic based notification channel for sending notifications. A topic is a subject of common interest among the services participating in a workflow, to which all notification messages of the workflow are sent. The publish/subscribe notification mechanism and the creation of a topic are transparent to end-



Generic Service Toolkit

Welcome to *BLAST* service

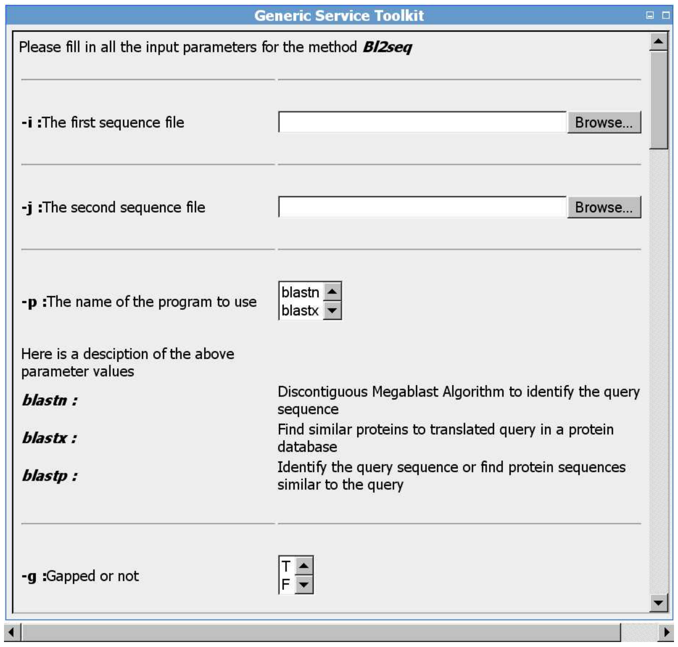
Service Description: This service will let you rapidly search a nucleotide or protein database

Please choose the method you want to invoke.

○ SearchDatabase	This method lets you search a protein or nucleotide database for the given DNA sequence
○ CreateDatabase	This method lets you create your own database using a FASTA formatted protein or nucleotide sequence. You can download FASTA formatted files from ftp://ftp.ncbi.nih.gov/blast/db/
○ PSIBLAST	This takes a protein query and performs PSI-BLAST search to creates a position specific matrix using a protein database
● Bl2seq	Bl2seq performs a comparison between two sequences using either the blastn or blastp algorithm. Both sequences must be either nucleotides or proteins

Go Back

Fig. 2. Service Interface

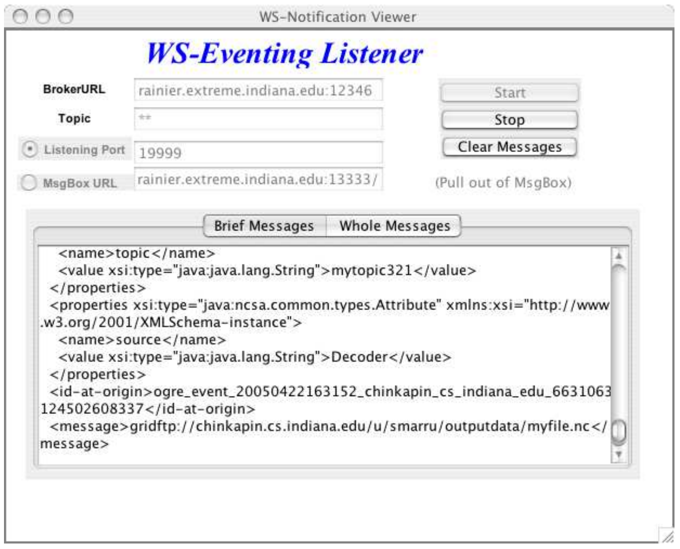


Generic Service Toolkit

Please fill in all the input parameters for the method ***Bl2seq***

-i :The first sequence file	Browse...
-j :The second sequence file	Browse...
-p :The name of the program to use	blastn / blastx
Here is a desciption of the above parameter values	
blastn :	Discontiguous Megablast Algorithm to identify the query sequence
blastx :	Find similar proteins to translated query in a protein database
blastp :	Identify the query sequence or find protein sequences similar to the query
-g :Gapped or not	T / F

Fig. 3. Method Interface



WS-Notification Viewer

WS-Eventing Listener

BrokerURL rainier.extreme.indiana.edu:12346

Topic **

● Listening Port 19999

○ MsgBox URL rainier.extreme.indiana.edu:13333/

Start Stop Clear Messages (Pull out of MsgBox)

Brief Messages | Whole Messages

```
   <name>topic</name>
   <value xsi:type="java:java.lang.String">mytopic321</value>
  </properties>
  <properties xsi:type="java:ncsa.common.types.Attribute" xmlns:xsi="http://www
.w3.org/2001/XMLSchema-instance">
   <name>source</name>
   <value xsi:type="java:java.lang.String">Decoder</value>
  </properties>
  <id-at-origin>ogre_event_20050422163152_chinkapin_cs_indiana_edu_6631063
124502608337</id-at-origin>
  <message>gridftp://chinkapin.cs.indiana.edu/u/smarru/outputdata/myfile.nc</
message>
```

Fig. 4. Notification Listener

users. There are three main components in our notification model; the Notification Consumer, the Notification Publisher, and the Notification Broker [23]. The Notification Consumer is an event sink. It is a Web Service that waits for notifications to arrive and handles them appropriately. All services that need to receive notifications use this service. The Workflow Execution Engine that executes a workflow subscribes to this service to receive notifications from all the services in the workflow. The Notification Publisher is used to publish notifications. All services created by the *Generic Factory Service* have a built-in Notification Publisher to publish notifications. The Notification Broker is a service that acts as an intermediary and relays messages from a publisher to a consumer. It is persistent in nature and stores messages that cannot be delivered to the consumer. These messages can be retrieved later by the consumer.

C. Architecture of an Application Service

The Factory creates an application service from its description in the ServiceMap document. Neither the application provider nor the Factory generate any code for implementing the service interface. Also, no client side stubs or server side skeletons are created. So how does the Factory convert a service description into an actual service? The answer lies in the *message processor* that is present in all application services created by the Factory. The *message processor* is a very simple Web Service. It receives SOAP request messages and returns SOAP response messages as all Web Services do. However, it cannot process the request messages because it does not know how to process them. The information that it needs to process the request messages is given to it by the application provider as a ServiceMap document. When given a ServiceMap document, the *message processor* reconfigures itself to support all the operations specified in the document. It then generates its WSDL and registers it with a Registry. Based on the information in the service's WSDL, a client can create a SOAP request message for the operation that the user wants to invoke on the service. When the *message processor* receives such a request message, it validates the request message and invokes the application associated with the operation, in a separate thread of execution. The mapping between the operation and the application is obtained from the ServiceMap document. The list of input parameter values for the operation has a one to one correspondence to the list of input parameter values for the application.

The Factory can invoke almost any command-line application like UNIX commands or more sophisticated scripts written in OGRE, Jython, Python, Perl etc. OGRE, Jython and Python scripts are executed within the application service's

Java Virtual Machine. This enables the service to monitor the status of these scripts and send notification messages to a Notification Service.

The design of the *message processor* is simple yet powerful. It makes the application service lightweight yet highly configurable. No code generation or code deployment is needed to create a service from its description. If the server side logic changes, the application provider needs to just upload the new ServiceMap document to the application service; there is no need to create a new application service or a new instance of the application service. As an interesting side note, the Factory itself is an application service that is capable of creating other application services. It has the same simple *message processor* and a ServiceMap document that specifies a single operation viz. createService. This operation takes as input a ServiceMap document and creates an application service from it. Thus the Factory and all other application services have the same architecture; a message processor that processes request messages according to the specifications in the ServiceMap document.

D. Security in Application Services

Figure 5 shows the interactions among various services in our framework. An application provider securely logs into the Portal and loads her Grid proxy certificate into her Portal context. She then uploads the ServiceMap document to the Factory, which uses her proxy certificate to start the service on the remote host using GRAM. After the service is instantiated, it registers its WSDL with a Registry. Then, based on the policy information in the ServiceMap document, the service creates capability tokens and registers them with the Capability Manager [24] Service. Capability tokens are created using XPOLA [24]. Conforming to the principle of least authority (POLA), XPOLA is a fine-grained authorization infrastructure for Web and Grid Services and is based on *capabilities*. Every capability token is a detailed policy document containing authorization information for the service instance and all its operations. It is signed by the application provider's proxy certificate. The whole infrastructure includes a persistent Capability Manager Service, plug-in capability handlers on the service and client sides and a portlet-based user interface (shown in Figure 6) for providers and users to manage their capability tokens. Enforcement plug-in handlers guarantee that a user's request can do no more than what it is allowed as specified in the assigned capability token. It is important to note that application providers and users rarely need to manage their capability tokens because the framework manages them on their behalf, as follows:

- The application services create the capability tokens from the policy information in the ServiceMap document and register them with the Capability Manager Service.
- The *Generic Service Portlet* contacts the Capability Manager Service and loads the user's capability tokens into her Portal context just before accessing any service.
- The application services renew their capability tokens after they expire. The renewal policy is specified in the ServiceMap document by the application provider.

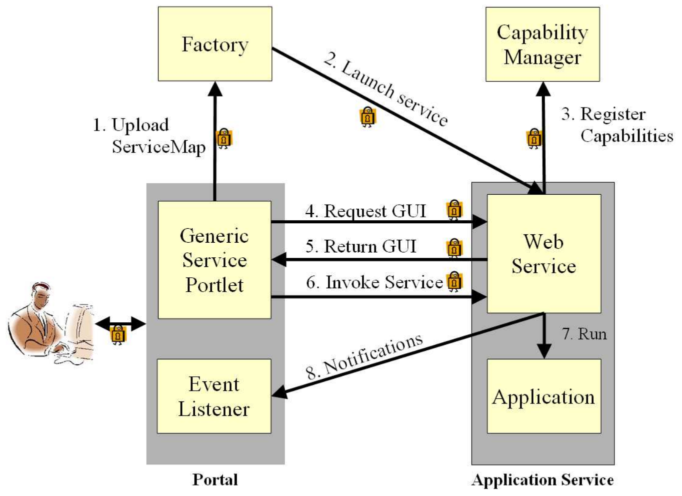


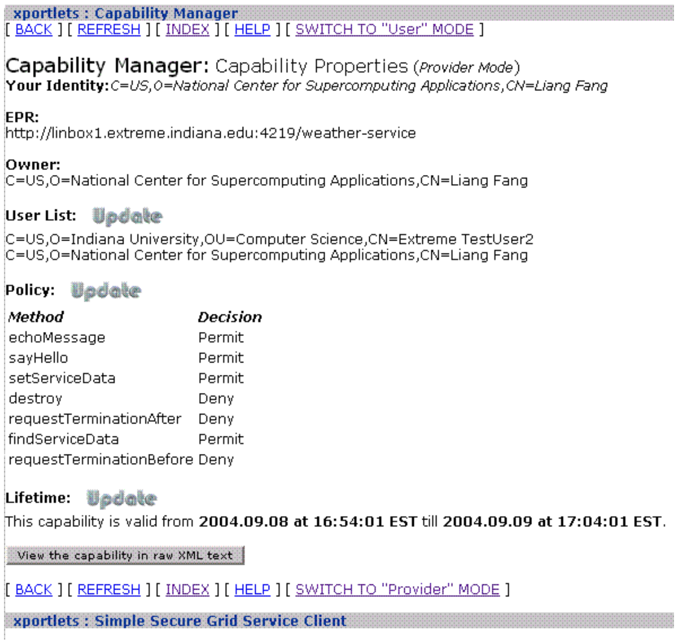
Fig. 5. Service Interactions

To access a service, a user logs into the Portal. She then uses the *Generic Service Portlet* to access the service. The portlet contacts the Capability Manager Service and loads the user's capability tokens into her Portal context. The portlet then sends a request to the service using SSL. The user's proxy certificate is used for authenticating the user to the service and the application provider's proxy certificate is used for authenticating the service to the user. After mutual authentication, the service returns the user interface. All further interactions with the service are secure and are done through the portlet using SSL. When the user invokes an operation on the service, the portlet sends a SOAP request message along with the user's capability token, signed by the user's proxy certificate. The service verifies the authenticity of the request and carries out the operation on behalf of the user, if the user is authorized to do so. While this protocol may seem potentially slow and complex, our initial experience is that it is neither slow nor is the complexity a problem because the framework handles it a manner that is transparent to application providers and users. A more detailed and quantitative analysis of the performance is underway and will be published later.

IV. Composing Workflows from the Portal

So far, we have described how to wrap a single application as a Web Service. However, it is common for scientists to execute a sequence of applications to get desired results. In the traditional way, several command-line applications are glued together by scripting languages such as OGRE, Jython, Perl etc. This works well if the application is not distributed on the Grid. In the case of complex applications that operate across a Grid, a more service-oriented system is needed. Our workflow composer, called X-Workflow Composer, enables users to graphically compose workflows from Web Services. It provides an easy-to-use GUI that allows users to search for interesting services, visually connect them together to form workflows and execute the workflows on the Grid.

As we have seen in section III-A, an application provider uses the ServiceMap document to describe her service. This document is converted to an abstract WSDL by our workflow



xportlets : Capability Manager
[BACK] [REFRESH] [INDEX] [HELP] [SWITCH TO "User" MODE]

Capability Manager: Capability Properties (*Provider Mode*)
Your Identity: *C=US,O=National Center for Supercomputing Applications,CN=Liang Fang*

EPR:
http://linbox1.extreme.indiana.edu:4219/weather-service

Owner:
C=US,O=National Center for Supercomputing Applications,CN=Liang Fang

User List: Update
C=US,O=Indiana University,OU=Computer Science,CN=Extreme TestUser2
C=US,O=National Center for Supercomputing Applications,CN=Liang Fang

Policy: Update

Method	*Decision*
echoMessage	Permit
sayHello	Permit
setServiceData	Permit
destroy	Deny
requestTerminationAfter	Deny
findServiceData	Permit
requestTerminationBefore	Deny

Lifetime: Update
This capability is valid from **2004.09.08 at 16:54:01 EST** till **2004.09.09 at 17:04:01 EST**.

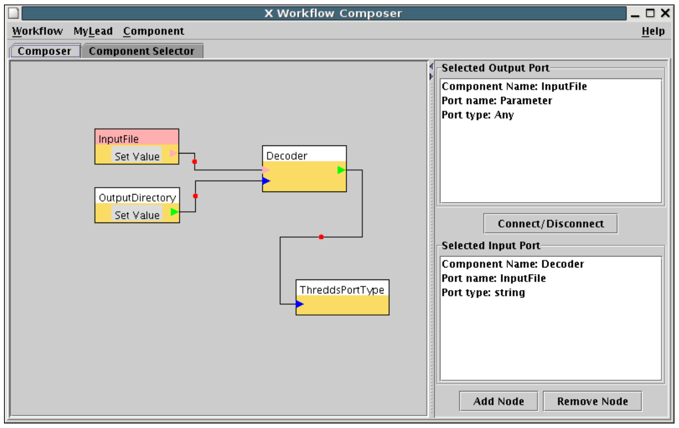
View the capability in raw XML text

[BACK] [REFRESH] [INDEX] [HELP] [SWITCH TO "Provider" MODE]
xportlets : Simple Secure Grid Service Client

Fig. 6. Capability Manager Portlet

composer and registered with a Registry. While a WSDL represents a service instance, an abstract WSDL represents a service. It allows the user to create workflows from non-existent services. These services can be instantiated dynamically by the Factory when the workflow is actually executed on the Grid. The abstract WSDL contains information about the port-types of the service, the operations and their input and output data types. Using the X-Workflow composer, the user first searches a Registry for interesting services. Each service is represented as a node with one or more inputs and outputs. The user creates a graph by interconnecting the services that constitute the workflow. The abstract WSDL also contains metadata about the service, port-types, operations and the input and output parameters. Some of this metadata is provided by the application provider in the ServiceMap document. Metadata about the input and output parameters can be obtained from a THREDDS [25] Catalogue Generator Service. In the next release of our composer, we plan to incorporate a plug-in module that can extract semantic information from the various types of metadata and assist the user in composing workflows. Figure 7 shows a snapshot of the X-Workflow Composer creating a workflow. After the user creates the graph that represents the workflow, the composer analyzes the dependencies among the constituent Web Services. It then creates a workflow script in Jython which can be executed by our Jython Workflow Engine. We are currently working on supporting workflow scripts in BPEL [26] which is a promising standard for describing workflows in the Web Services world.

A particular abstract application service may have several concrete instances running at the same time on the Grid. Each service instance may have a different policy associated with it. For example, services of a particular scientific community may not allow users of other communities to access them. The services created by our *Generic Factory Service* include this policy information in their WSDLs. Before executing the workflow, our Workflow Engine searches the Registry for service instances that satisfy the policy requirements. Our current Workflow Engine requires all services in the workflow to be running at the time of invocation. We will overcome this limitation when we replace Jython workflow scripts with BPEL workflow scripts and use a BPEL Workflow Engine that can dynamically create service instances using our Factory. After starting the workflow, the Workflow Engine receives notification messages from the services to monitor their status. Notification messages contain status information and output parameters. The Workflow Engine may use the output parameters of a service as the input parameters to the next service in the workflow. It also monitors the services for error messages. In the event of a failure of a service invocation, the entire workflow is stopped. In our next version where we plan to use BPEL scripts, the BPEL Workflow Engine will be able to handle failures more gracefully.



Selected Output Port
Component Name: InputFile
Port name: Parameter
Port type: Any

Selected Input Port
Component Name: Decoder
Port name: InputFile
Port type: string

Fig. 7. Workflow Composer

V. Conclusions

Our generic framework allows scientists to wrap applications as Web Services, compose workflows from them and execute them on the Grid. The services created by our framework generate their own graphical user interface, which allows end-users to interact with them using thin and generic Web Service clients. Security is one of the prime concerns in a distributed environment. Our framework takes care of authentication and authorization during all client-service interactions in a manner that is almost transparent to application developers and end-users. Users can search for services and compose workflows by visually interconnecting them in a workflow composer. Services use notification messages to report their status and results. Using this framework we have been able to integrate some real scientific applications into distributed Grid applications based on Web Service architecture.

References

[1] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, "Web Services Description Language (WSDL) 1.1," 15 Mar. 2001; http://www.w3.org/TR/wsdl

[2] D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan, and A. Slominski, "Grid Web Services and Application Factories," *Grid Computing: Making the Global Infrastructure a Reality*. Fox, Berman and Hey, eds., Wiley, 2003.

[3] M. Senger, "Soalab: SOAP based analysis web service," 24 Feb. 2005; http://industry.ebi.ac.uk/soaplab

[4] M. Senger, P. Rice, and T. Oinn, "Soaplab - a unified Sesame door to analysis tools," *Proceedings of the UK e-Science All Hands Meeting,* 2-4 Sep. 2003

[5] "Axis," 10 Apr. 2005; http://ws.apache.org/axis

[6] "CORBA/IIOP specification," 04 Mar. 2001; http://www.omg.org/technology/documents/formal/corba_iiop.html

[7] M. Senger, "Gowlab: Web pages as web services," 3 Mar. 2005; http://industry.ebi.ac.uk/soaplab/Gowlab.html

[8] C. Letondal, "PISE: A tool to generate web interfaces for molecular Biology programs," 10 Dec. 2004; http://www.pasteur.fr/recherche/unites/sis/Pise

[9] "GridLab products and technologies," 6 Apr. 2005; http://www.gridlab.org/about.html

[10] "Grid(Lab) Grid Application Toolkit," 30 Jun. 2004; http://www.gridlab.org/WorkPackages/wp-1

[11] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, Y. Zhao, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows,* to be published 2005

[12] "SeqHound," 10 Apr. 2005; http://www.blueprint.org/seqhound

[13] "BioMoby,"; http://biomoby.org

[14] "Kegg," 10 Apr. 2005; http://www.genome.jp/kegg

[15] S. Hampton and A.L. Rossi, "OGRE: Programmer's manual," 10 Apr. 2005; http://corvo.ncsa.uiuc.edu/ogre/docs/manual/index.html

[16] "JSR 168 portlet specification (final release)," 7 Oct. 2003; http://www.jcp.org/aboutJava/communityprocess/final/jsr168

[17] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, 1997, pp. 115-128

[18] "GT3 GRAM architecture," 4 Jun. 2003; http://www-unix.globus.org/developer/gram-architecture.html

[19] A. Kropp, C. Leue, and R. Thompson, "Web Services for Remote Portlets specification," 9 Mar. 2003; http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf

[20] "The GridFTP protocol software," 27 Sep. 2002; http://www.globus.org/datagrid/gridftp.html

[21] S. Graham et al., "Web Services Base Notification version 1.0," 5 Mar. 2004; ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BaseN.pdf

[22] D. Box et al., "Web Services Eventing (WS-Eventing)," Aug. 2004; ftp://www6.software.ibm.com/software/developer/library/ws-eventing/WS-Eventing.pdf

[23] S. Graham et al., "Web Services Brokered Notification version 1.0," 5 Mar. 2004; ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BrokeredN.pdf

[24] L. Fang, D. Gannon, and F. Siebenlist, "XPOLA: An extensible capability-based authorization infrastructure for grids," *4th Annual PKI R&D Workshop: Multiple Paths to Trust,* 19-21 Apr. 2005

[25] B. Domenico, J. Caron, E. Davis, R. Kambic, and S. Nativi, "Thematic Real-time Environmental Distributed Data Services (THREDDS): Incorporating Interactive Analysis Tools into NSDL," *Journal of Digital Information*, vol. 2, no. 4, 2002.

[26] T. Andrews et al., "Business Process Execution Language for Web Services version 1.1," 5 May. 2003; ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf