Cyberaide JavaScript: A JavaScript Commodity Grid Kit

Gregor von Laszewski[1], Fugang Wang[1], Andrew Younge[1], Xi He[1], Zhenhua Guo[2], Marlon Pierce[2]

[1]**Rochester Institute Of Technology, 102 Lomb Memorial Drive Rochester, NY 14623**

[2]**Community Grid Lab, Indiana University, Bloomington, IN 47404**

E-mail: laszewski@gmail.com

Contents

Cyberaide JavaScript: A JavaScript Commodity Grid Kit

Gregor von Laszewski[1], Fugang Wang[1], Andrew Younge[1], Xi He[1], Zhenhua Guo[2], Marlon Pierce[2]
[1]Rochester Institute Of Technology, 102 Lomb Memorial Drive Rochester, NY 14623
[2]Community Grid Lab, Indiana University, Bloomington, IN 47404
E-mail: laszewski@gmail.com

Abstract

In this paper, we describe a service oriented architecture and Grid abstraction framework that allows us to access Grids through JavaScript. Obviously, such a framework integrates well with other Web 2.0 technologies. The framework consists of two parts. A client Application Programming Interface (API) to access the Grid via JavaScript and a mediator service and API through which the Grid access is channeled. The framework uses commodity Web service standards and provides extended functionality such as asynchronous task management, file transfer, and workflow management based on our previous work. The availability of our framework simplifies not only the development of new services, but also the development of advanced client side Grid applications that can be accessed through Web browsers. We demonstrate this ability by providing an example that integrates a variety of useful services to be accessed through a JavaScript enabled client desktop via a Web browser. Overall, Grid developers will have another tool at their disposal that projects a simpler way to distribute and maintain cyberinfrastructure related software, while simultaneously delivering advanced interfaces and integrating social services for the scientific community.

Keywords: JavaScript, Grid Computing, Workflow, CoG Kit, Grid Abstractions.

1 Introduction

In this paper, we describe a Service Oriented Architecture (SOA) [1] and Grid abstraction framework that allows us to access Grids through JavaScript. It is based upon abstractions that have proven to be useful in the Grid community and are the result of research activities from the Java CoG Kit [2]. Nevertheless, we are enhancing the previous approach with a number of advanced services as well as targeting JavaScript as the language of choice for the client to support Web 2.0 style portals. Through the framework we can integrate with a variety of Grid middleware and obtain access to the Grid fabric. In this paper we focus primarily on the integration with Globus [3] and access to the TeraGrid [4]. The framework contains a useful set of JavaScript functions to simplify this access.

One of the advantages of the JavaScript API is we can integrate a large number of commodity libraries available in JavaScript to go beyond the traditional use of Grid technologies. Hence, we are able to leverage from data-structure libraries, social networking and communication libraries to enable Web 2.0 programming features and allow access to additional commodity cyberinfrastructure that would otherwise be difficult to be achieved. As a result the framework will be more than just a Grid client library. In order to emphasize this difference and its ability to function as an aide for integrating cyberinfrastructure in general, we use the term *Cyberaide Javascript.*

2 Background and Related Research

Grid portal development has been going on for quite some time. The first usable library to support the development of Gridportals was the Java CoG Kit [5]. Based on the premise of the Java write-once-run-everywhere concept the Java CoG Kit was designed explicitly to be a 100% Java-based library. Originally, Applets were developed that were soon enhanced by JSR168 [6] compatible portlets. These portlets were then integrated and enhanced as part of the Open Grid Computing Environments (OGCE) Project [7] and by Gridsphere [8]. Together these three projects

(CoG Kit, OGCE, and Gridsphere[1]) build a major foundation for the TeraGrid portal [9], which is one of the premier NSF sponsored resource in the U.S. to obtain access to Grid resources. TeraGrid uses the Globus Toolkit [3] to manage its Grid resources.

However, the current generation of Web based technologies have integrated JavaScript as one of the major offerings. Unfortunately, to date no JavaScript framework exists that lets us access the Grid easily.

While other Portal and Web technologies exist, we do not list them as they are beyond the scope of this paper.

3 Design

In order to design a JavaScript library for the Grid we identified the following requirements as an essential feature:

- *Ease of use* is important to make the JavaScript based API and interfaces useful for Grid and Web developers.
- *Low installation footprint* is necessary to support fast downloads as well as an easy maintenance through a small manageable code base.
- *Security* is needed to gain access to Grid resources in order to avoid compromising the system. This is especially important due to known limitations of JavaScript.
- *Basic Grid functionality* must be provided in order for developers to create Grid-based client applications.
- *Advanced functionality* is needed as many developers do not want to replicate functionality provided by other Grid middleware and upperware. This includes more sophisticated job management functionality, workflow queues, and the availability of elementary graphical user interfaces (GUI)s.

To fulfill these requirements, we have designed our framework methodologically as a layered architecture. The architecture is comprised of several components. We depict the most important components of our architecture in Figure 1 and explain them in more detail next. The architecture contains a Web client providing access to basic and advanced Grid functionality, as well as the necessary components to deploy them in a secure Web server. To keep the footprint of the library small most of the functionalities in regards to the Grid are executed on a secure server. As this server mediates the tasks to the Grid, we refer to it as the *mediator service*.

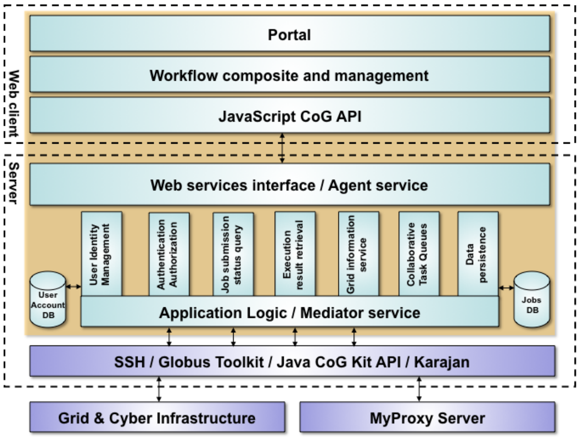

Web client
Portal
Workflow composite and management
JavaScript CoG API
Server
Web services interface / Agent service
User Identity Management
Authentication Authorization
Job submission status query
Execution result retrieval
Grid information service
Collaborative Task Queues
Data persistence
User Account DB
Jobs DB
Application Logic / Mediator service
SSH / Globus Toolkit / Java CoG Kit API / Karajan
Grid & Cyber Infrastructure
MyProxy Server

Figure 1: System Architecture

First, our Web client provides a high-level application programming interfaces (API) to the Grid, a Grid workflow system, and components to access simple Grid functionality through graphical user interfaces. While using the API the developers can build portals specifically targeted for Grids and integrate customized JavaScript-based GUIs. Furthermore, the framework provides important functionalities such as authentication, file transfer, job management, and workflow management.

Second, we are conducting all interactions to the backend Grid or cyberinfrastructure through the mediator service. The mediator service is responsible for communicating with Grid services through underlying Grid middleware such as the Java CoG Kit, Globus toolkit, or even SSH. Meanwhile it exposes the Grid services via standard Web services. It allows the use of a personal queue management mechanism that keeps track of all jobs and workflows submitted to the Grid in a more convenient form than the current generation of Grid middleware, such as the Globus Toolkit. The mediator service contains several essential functional modules such as user account management, authentication, job management, collaboration queues, and a persistence module allowing users to keep track of their jobs and tasks on the Grid.

[1] Most recently the decision was made to replace Gridsphere with Websphere

An intermediate component exists called the *agent* service that handles all communications between the web client and the mediator service. This means the agent service acts both as a service provider to the web client and as a service consumer of the mediator service. The agent service forwards the action requests to the mediator service, which ultimately sends requests to the Grid services. The results come back from the Grid through the mediator to the agent, which in turn forwards the information back to the client. We have to host the JavaScript files, CSS files and other resources in some web container, and according to the same-origin policy for JavaScript [10], we need to put the web service that the JavaScript calls in the same web container. By separating the real application logic and putting it into the mediator service, it is possible to host the services on different machines, which would increase security and scalability.

3.1 Security Considerations

Naturally, security is of utmost importance in the development of any portal framework to the Grid. It is important to identify possible security issues that may arise in a JavaScript-based solution.

Since HTTP is stateless, we need to maintain some method to record users' states. Typically this is done by using HTTP cookies [11] which may include user and/or session related information. However, due to the well-known Cross-site Scripting(XSS) [12,13] and Cross-site Request Forge (CSRF/XSRF) [14] vulnerabilities, we avoid the use of cookies to minimize the potential risk of these attacks. Instead we use a security token that is assigned to an authenticated user in order to prove users' identity during the session.

Security tokens maintain session information in similar way to cookies. Since we maintain the token only inside a JavaScript object during a session, it is immune to cookie related attacks, which typically access the cookie using *document.cookie* [15,16].

The security architecture is supported by the following features.

- HTTP over SSL/TLS (HTTPS) [17] is used between the web client and web server to gain transport layer security.

- WS-Security standard [18] is used to secure the Web service traffic between the web server and the underlying mediator service.

- Grid Security Infrastructure (GSI) [19] is used between the mediator service and the Grid.

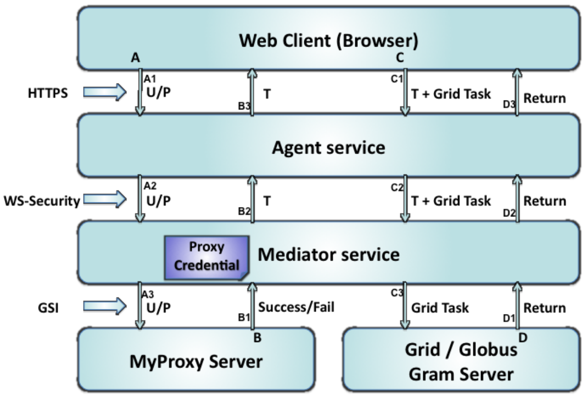


Figure 2: Security view of a typical use scenario

To better understand some of the security aspects we walk through the following scenario where a user authenticates using MyProxy and performs tasks such as job submission, status query and result retrieval. In Figure 2 flows are corresponding to (A) user authentication request, (B) authentication response, (C) Grid related actions request and (D) Grid responses. A flow consists of multiple steps that are indicated by the inclusion of a number for the step.

1. An *authentication request* is shown by flow (A). A user tries to login to the system by providing the username/password (U/P) in the web client as in step (A_1). HTTPS guarantees that the traffic is secured between the browser and the agent service. The agent service then forwards the user credential to the user authentication module in the application logic layer (A_2). The authentication module could be used together with a user account management module or an external federated identity system such as MyProxy as in step (A_3).

2. An *authentication Response* of the authentication request from server is conducted as part of Flow (B). If the user provided a valid credential, then a security token (T) is returned to the agent service from the mediator service, as in step (B_2). The security token finally returns to the web client, which is depicted by step (B_3).

The token is generated randomly each time. The authenticated username and corresponding security token are maintained in a map so the token could be used later to prove the user's identity and also function as a session ID. This mechanism is similar with what is used in HTTP cookies. Implementing a cookie solution is avoided to eliminate the cookie related security vulnerabilities.

3. *Grid related actions* such as the submission of jobs, status queries, or retrieval of results are represented by flow (C). The client JavaScript API will append the security token to the user's request to the agent service as shown in step (C_1) to prove the user's identity during the current session. Next the agent service could forward the task appended with the token to the mediator service, as shown in step (C_2). Then the token is checked for validity in the mediator service. If it is valid, the mediator service will invoke Grid services as desired as shown in step (C_3). When the user logs out of the system or the session duration is expired, the security token will be invalided and recycled by the Mediator service and no further actions will be carried on to the underlying Grid services.

4. *Grid responses* that occure during task submission are shown in flow (D). The responses of the Grid related action requests are forwarded all the way back up to the client, which is shown in steps $(D_1) through (D_3)$.

3.2 Web Client

The Web client provides the elementary functionality to access the Grid through a portal user interface. The portal fulfills the tasks by using the JavaScript API allowing access to a number of essential Grid services. These services provide the following functionality to the user:

1. Creation of jobs and workflows on the client side;

2. Submission of jobs for execution or for inclusion in shared queues that are managed by multiple users.

3. Information queries and monitoring of the status of jobs and workflows.

It is important to note that at this time Globus GRAM does not support managing jobs by multiple users. We are able to provide this functionality because all user jobs are managed through the mediator service that allows jobs to be executed through a service in behalf of a coordinating user.

3.3 Server

As described before, the server functionality contains two logical parts, the Web services interface also referred to as user agent service and the Mediator service that builds the bridge between the Grid and our client library.

3.3.1 Agent service

The *agent service* functions as an intermediate service between the Web client and the mediator service. It hides much of the complexity of the Grid and allows for the deployment and integration of resources where the installation of Grid middleware is not possible. Hence, it works as a proxy for users to interact with the mediator service, and through it to Grid services. It can perform proxy credential delegation for users to authenticate to the Grid and to provide single sign on. The agent uses web services to provide functionality calls and communication with web client. Thus, the agent service itself is a service consumer of the mediator service.

3.3.2 Mediator Service

The *mediator service* is where the application logic code resides. It contains several modules to deal with different functionalities and offers a persistent view of interactions with regular Web services, Grid services, or authentication. While the agent service just forwards all requests from the user to the mediator service, the mediator service supports multi-user concurrency by maintaining session and state for each user. A persistent database is used to maintain state about workflows and collaboratively managed queues. It is important to recognize that our notion of compute tasks exceeds that of the basic Globus functionality. It also allows one to integrate services into the tasks managed by a user that are not typically executed by Globus. Thus a query to Web services offered by another organization like Google can be readily abstracted a task with its own status. Hence, our task model includes not just tasks that are executed on Globus enabled Web services.

3.4 Other Client Interfaces

Due to the separation between the service and the client, it was also possible to develop a *Cyberaide shell* [20] that allows access to the service functionality from a command line console. Of special interest for the user is the ability to have semantically enhanced commands as part of this shell. Thus, a command such as

```
submit /home/gregor/myprg -notify Gregor
```

would look up an object called Gregor that specifies where to send the notification to. This may not just be an e-mail. The user may decide to have notifications forwarded to his cell phone via SMS messaging. The behavior is determined during runtime.

4 Implementation

The system design is based on object-oriented model and Service-Oriented Architecture (SOA) [21–23]. Entities in the system are all objects, while a SOA is used for the interaction between the distributed objects. The benefit of this approach is that it enables a loose coupling between distributed objects. Thus, our concern is how to manage contracts between services and not the actual implementation details of the objects that may be implemented in different languages and frameworks. As long as we keep the contract between entities, changes on implementation of one part would not affect other parts of the system.

Although JavaScript is not a strong typed language and does not provide a general mechanism for defining a class or object-type definition [24], it is a well-established practice to define custom objects in JavaScript that behave, in many ways like classes in Java. Hence we will use the term JavaScript classes throughout the text.

4.1 Web Client

One of the most important feature for many users will be the definition of APIs in JavaScript dealing with job files, and security management. Based on the lessons learned from the Java CoG Kit, we designed APIs in JavaScript that address the well-established functionality needed by many users [25]. This includes tasks that manifest themselves as authentication, file transfers, jobs, status queries, workflows [26], and experiments [27].

4.2 Application Programming Interface

Table 1 lists a small subset of the functionality that we are currently implementing. Built upon the JavaScript CoG APIs, the portal provides generic purpose functionalities for access Grid service while communicating with the mediator. While we use SOAP as message format between web client and agent service, the output of the JavaScript APIs is transformed into JSON [28, 29] format. The return values are handled as part of callback functions that can be customized by the developers using our API. However, we provide a significant number of useful callback handlers as part of or documentation to demonstrate to serve as template for customization.[2] This also includes a comprehensive set of documented JSON objects that will make the development of customized callback functions easier. Thus, it allows us to easily integrate data into the JavaScript client program. External JavaScript libraries such as DOJO toolkit [30] can be used to ease the development of a user friendly graphical interface with desktop application comparable performance so we don't need to reinvent wheels. An example interface of the services are listed in Section 4.6.

We have divided the application interface into multiple categories. These include JavaScript objects dealing with jobs, workflows, and authentication. In addition we provide many functions that are used as part of workflow management, job management, and queue management. For methods with callback function in their definition we can also call them without a callback function. In this case, the function will be executed synchronously, waiting for its completion and returning a JSON object.

In asynchronous mode a user can pass a predefinedcallback function as parameter to a method call. The callback function uses a single JSON object as parameter and will be automatically called when a response is available. Alternatively a user could use a function to call a method simply by omitting the callback parameter. In this mode the method will return a JSON object that the user can operate on synchronously and wait till the response is available.

To demonstrate the use of callbacks please review Figures 3-6. Figure 3 shows an example which defines a CoGExecutable object. Figure 4 and Figure 5 show the two different modes to call and use an

[2] To assist other developers, we are exploring the creation of a community repository in which users can share their templates for handling cyberinfrastructure related patterns.

Table 1: Application interface

Class	CoGExecutable – functions related to Job and Workflow Abstraction	
		Object with a number of attributes that abstracts the concept of execution of jobs.
Class	**CoGAuthenticator – functions related to Security**	
○	authenticate (callback)	Authenticate through the attributes by using the provider defined in the CoGAuthenticator object.
Class	**CoGBase – functions related to User Job Management**	
○	authenticate(CoGAuthenticator, callback)	The CoGAuthenticator's authenticate() function will be called. For example, if we use in the attribute provider as part of the CoGAuthenticator class attributes the value MYPROXY, then a myproxy authentication will be carried out.
○	submit(CoGExecutable, callback)	Submit an Executable to server for asynchronous execution.
○	transfer(source, destination, callback)	transfer data from source to the destination each of which are defined as URI. A transfer itself is packaged as a submit function and is treated the same way as other executables are.
○	list(callback)	List all the CoGExecutables submitted to the server.
○	queryStatus(executablesids, callback)	Get state of the set of executables specified by their ids.
○	queryOutput(executableids, callback)	Get list of all the output files the set of executables specified by their ids.
○	getOutput(executableids, resultFile, callback)	Get back the content of one output file of the specified executable ids.
Class	**CoGWorkflow – functions related to Client Side Workflow Management**	
○	addJob (name, CoGExecutable)	A new job with a given name is added to this workflow.
○	deleteJob (name)	The job with the name specified by parameter job name is deleted. As a result, all related dependency is deleted as well.
○	listJobs( )	List all jobs in a workflow.
○	searchByName(job)	Search a job by name.
○	addDependency (parent, child)	Add dependency between a job patent and the child.
○	removeDependency(parent, child)	Remove dependency between a job parent and child.
Class	**CoGQueue – Server Side Shared Queue Management**	
○	listQueues(callback)	List all the queues that the user are participating.
○	grantAccess(queueID, userlist, callback)	The owner of a queue can give some other users access privilege to the queue.
○	add(queuename, CoGWorkflow, callback)	adds a workflow to the queue to be executed.
○	remove(queueID, workflowID, callback)	The owner of a workflow can remove it.
○	list(queueID, callback)	list all workflows' metadata shared in the queue.
○	listParticipants(queueID, callback)	List all the participants of the queue.
○	listByName(queueID, username, callback)	List the workflows' metadata owned by a user from one queue.
○	listByType(queueID, provider, callback)	List all the workflows' metadata with the specified type, say, Karajan workflow, from the specified queue.
○	getStatus(queueID, workflowID, callback)	Query the specified workflow's status. It could be locked since being edited by a user.
○	browseWorkflow(queueID, workflowID, callback)	Download the workflow to client side to browse.
○	obtainWriteToken(queueID, workflowID, callback)	The user try to obtain the write token.
○	editWorkflow(queueID, workflowID, callback)	The user will try to obtain the write token and then to download the workflow to edit.
○	updateWorkflow(queueID, workflowID, WFObj, callback)	Update the modified workflow when complete editing.

API functionality either through an asynchronous or through a synchronous call. Figure 6 shows the returned JSON object format for the synchronous and asynchronous examples depicted in Figures 3 and 4

4.2.1 Job and Workflow Abstraction

The CoGBase class uses the above classes to perform the interactions with the Grid services, through the agent service and Mediator service. The callback functions has a single parameter *jsonRet*, which contains a JSON [28,29] object containing the response. To avoid security issues this response is wrapped into a single object as discussed in [31–33].

Based on the discussion in the previous section it is clear that the CoGExecutable class is used to define an object that will be executed in the Grid, which could be a single job or a number of jobs part of a larger workflow. In summary, the class has two important fields: a *provider* and *attributes*. A provider could have the values *system*, *cog-karajan* or any other value referring to a workflow engine assuming that support for the workflow type exists within the gridshell. Attributes describe properties of a task to be executed.

4.2.2 Security

The CoGAuthenticator class is used to define objects that handle the authentication in the JavaScript. It has a *provider* and *attributes* fields and uses an authenticate() function. The provider represents the authentication method to be used. Currently we support *username-token* and *MyProxy* authentication. The attributes contain the necessary information to authenticate the user while using the method provided in the Provider field. For username-token authentication, it would contain the username and password/token. For MyProxy authentication, the attributes should contain the MyProxy server's hostname, port, username and corresponding passphrase to retrieve the proxy credential. For the current prototype system, MyProxy provider is used as the default authentication mechanism. The authenticate function deals with initiating the authentication with the server.

4.3 Agent Service

We have implemented the agent service with the help of standard Web service technologies. Hence, SOAP [34] messages are used for communicating between

```
// Example to define a single remote
// job and submit it:
var jobA = new CoGExecutable();
jobA.provider = "system";
var attribs = {};
// specify a command and arguments
attributes.command = "/bin/ls";
attributes.arguments = "-l";
//redirect standard output into stdout
attributes.stdout = "stdout" ;
//specify the remote machine and service
//.e.g. GT4
attributes.host = "fast.rit.edu";
attributes.globusProviderVer= "GT4" ;
...
```

Figure 3: Example of defining a CoGExecutable object.

```
// Now you can submit the job
// through a CoGBase object in an
// asynchronous way. You need to
// define a callback function for the
// asynchronous call, which will be called
// automatically when a JSON
// object containing the jobId is returned

var cog = new CoGBase(url)
// url specifies the web service endpoint
cog.submit(jobA, callback1);
...
// pre-defined callback funtion by the user
function callback1(jsonRet){
  ...
// do customized operations here, e.g., display
// the returned JobId in a textbox in the HTML page
}
```

Figure 4: Use asynchronous mode to invoke a method.

```
// you also call use synchronous call and
// wait until the json object returns.
var myJobId = cog.submit(jobA);
```

Figure 5: Use asynchronous mode to invoke a method.

```
{"wfid" : id}
```

Figure 6: Format of the returned JSON object for the above examples.

the agent service and web client. The connection is secured through HTTP over SSL/TLS.

Presently, we use Java to develop and deploy the Web services. We used JAX-WS [35] annotations from Java SE 6 to specify the service.

The service archive [36] is deployed under Axis2 [37] service with Apache Tomcat [38] web server.

To interact with the mediator service, the client stubs of the mediator service are generated by tools such as wsimport. The build process is controlled by maven [39] to allow for easy deployment and upgradability. All Web page resources including static web pages, CSS files, the JavaScript CoG library, Portal JavaScript files, and necessary external JavaScript libraries and images are hosted within the same Tomcat server under the same HTTPS host name and port.

Secure traffic between agent service and web client is obtained through HTTPS connection, while WS-Security is used to secure the mediator service.

4.4 Mediator service

In the implementation of the mediator that interacts with the Grid, we use the Java CoG Kit API and command line tools. Thus, through the Java CoG Kit we can enable proxy credential retrieval and delegation. In more detail, the Java CoG Kit JGlobus module provides functionalities to handle the interaction with MyProxy [40]. As long as a user stored his/her credential on a MyProxy server, the user can retrieve the credential by providing the username and user phrase at time when the proxy credential is generated. Thus the user delegates the application server to communicate with the underlying Grid infrastructure during the user session without the need to authenticate each time a job is submitted or a status query is issued.

To expose this service to the client, it is implemented and deployed as web service. WS-Security is used to secure the service traffic, either by UsernameToken mechanism or PKI mechanism. Apache CXF framework [41] is used to develop and secure the mediator service due to its simplicity. WS-Security is supportted in the CXF framework, and is used to secure the service traffic, either by UsernameToken mechanism or PKI mechanism.

User account info is maintained in a persistent database. However it would be possible to replace this mechanism with a federated authentication based on Shiboleth [42] to integrate external identity management systems. Tools and APIs from the Java CoG Kit and the Globus toolkit are used to establish the interaction with Grid services. Furthermore, the Java CoG Kit Karajan workflow framework is used to support workflow composition.

4.5 Collaborative Queue

Besides the typical Grid functionalities discussed in the previous section, our design includes features that enable users to share and manage workflows among a group of users collaboratively. As such we have defied a *shared workflow* and a *shared queue* that can contain multiple workflows to be executed. Through the concept of sharing we provide user-based access control supported by ownership and group based access control supported by membership in a participant list.

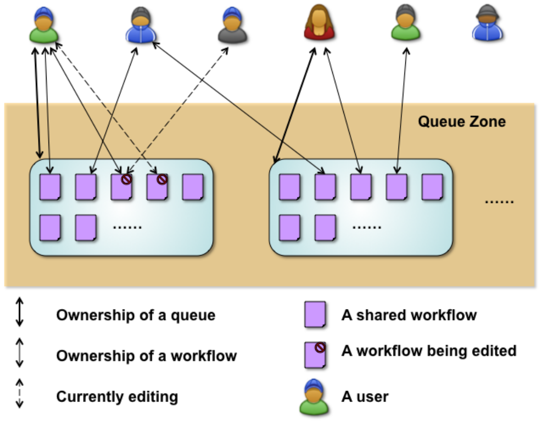


Figure 7: A scenario for a shared Queue.

To implement this functionality, we have defined the objects representing a shared queue and a shared workflow:

$$queue \quad ::= \quad (queueID, label, owner, participants, objects)$$

$$workflow \quad ::= \quad (workflowID, label, owner, lastmodbyuser, lastmoddate, writeToken, type, WFObj),$$

The attributes to these objects are as follows:

- *label* is a field to store a meaningful label to identify the object easily.
- *QueueID* is a unique ID for the queue.
- *owner* for the *queue* is the one who created the queue. Only the owner can grant other users

access to the queue. Ownership can be transitioned.

- *participants* contains a list of participants that are allowed to modify the queue.
- *objects* represent workflows managed by this queue.
- *workflowID* is a unique id for the shared workflow.
- *lastmodbyuser* is the last user who modified this queue.
- *lastmoddate* is the last time when it is modified by a user.
- *writeToken* is a lock to guarantee atomic write operation of the workflow object.
- *type* is the object's type. It could be system executable script or Karajan workflow, a Globus job, a file transfer, to name only a view.
- *WFObj* is the workflow description wrapped in this workflow object.

It is obvious from the definition of the attributes to that it will be possible to provide a shared workflow. As pointed out before, it is important for the activities typically conducted within an ad-hoc virtual organization to coordinate computational experiments as part of group activities. To support the notation of Workflows we have introduced a number of supporting APIs that makes the development of workflow related tasks easy for clients and services.

4.5.1 Client side Workflow Management

The Workflow class (see Table 1) represents a *CoG Karajan* workflow. A *CoG Karajan* workflow can contain multiple jobs, and it supports hierarchical workflow, which means a workflow could be a *job* in another workflow. A user can manage jobs with dependencies between them as part of a single workflow.

4.5.2 Shared Workflow Queue Management

The CoGQueue class (see Table 1) is used to define objects that deal with the interactions between the web client and the server side's collaborative queue management functionality. Users are able to store workflows composed on client side to the server side shared queue zone through a CoGQueue object. These workflows can also be managed and shared between users. Callback functions and the returned JSON objects share the same arguments as that in the CoGBase.

4.6 Graphical User Interface

We have developed a prototype of the system that handles tasks such as job and workflow specification, file transfers, status and information queries. This includes the development of a simple GUI supporting these tasks (see Figures 8-13). The prototype utilizes the Java CoG Kit as intermediary services; uses Axis2 [37] and Tomcat to develop and deploy the agent service; and uses Apache CXF framework [41] to develop and secure the mediator service.

User authentication is typically the first task. An external MyProxy authentication is used for user authentication. By providing appropriate MyProxy server settings and the right username and passphrase, users will be able to authenticate as shown in Figure 8.

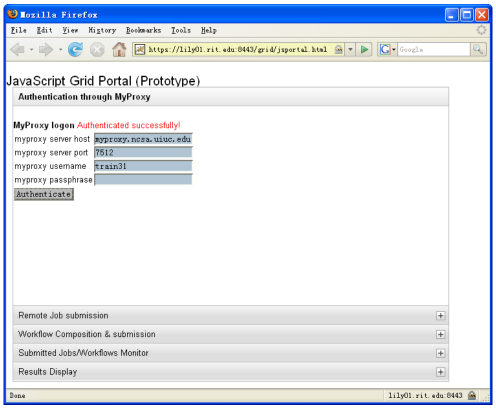
The *Histrory* of jobs allows us to check the history of submitted jobs/workflows as shown in Figure 9.

Job construction and workflow composition is supported by GUI for single job submission (see Figure 10) and for workflows (see Figure 11). For simple job specification users can fill out a form. For workflows we provide at this time a simple text window that accepts Java CoG Kit Karajan workflows.

Job execution is conducted once the job or workflow is specified. While explicitly pressing the submit button, the web client will invoke a number of web services through our mediator. Then a jobId will be returned to the client side as a handler. The execution of jobs can be monitored through a simple GUI as depicted in Figure 12.

Status query and results retrieval allows users can to check the status of the submitted jobs and workflows by clicking on the job/workflow id. Once all tasks specified within a workflow have been completed that status of the workflow is changed to *finished.* The result is recorded and can be viewed in the GUI (see Figure 13).

We also developed another version of the portal as in Figure 14, which also used the JavaScript APIs, to show the reusability of the API and diversity of the portal. Naturally these are just some elementary GUIs and we hope that more sophisticated one will be developed in near future. We encourage for example the community to contribute their components


Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://lily01.rit.edu:8443/grid/jsportal.html
Google
Authenticate
Remote Job submission
Workflow Composition & submission
Submitted Jobs/Workflows Monitor
Results Display
Done
lily01.rit.edu:8443

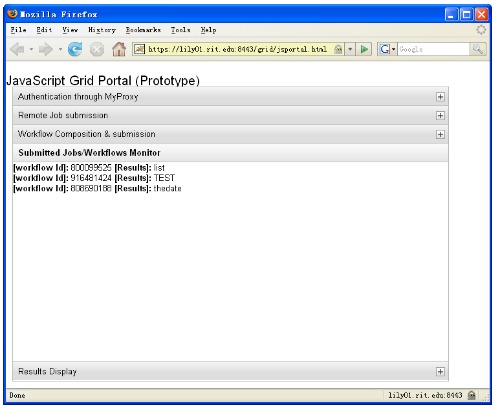
JavaScript Grid Portal (Prototype)

Authentication through MyProxy

MyProxy logon Authenticated successfully!

myproxy server host myproxy.ncsa.uiuc.edu
myproxy server port 7512
myproxy username train31
myproxy passphrase

Figure 8: User login page

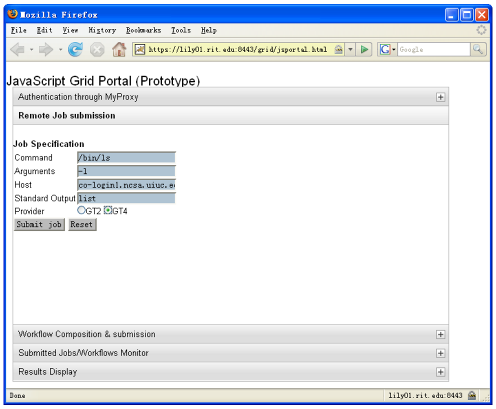

Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://lily01.rit.edu:8443/grid/jsportal.html
Google
Authentication through MyProxy
Remote Job submission
Workflow Composition & submission
Results Display
Done
lily01.rit.edu:8443

JavaScript Grid Portal (Prototype)

Submitted Jobs/Workflows Monitor

[workflow Id]: 800099525 **[Results]:** list
[workflow Id]: 916481424 **[Results]:** TEST
[workflow Id]: 808690188 **[Results]:** thedate

Figure 9: History of jobs


Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://lily01.rit.edu:8443/grid/jsportal.html
Google
Authentication through MyProxy
Submit job
Reset
Workflow Composition & submission
Submitted Jobs/Workflows Monitor
Results Display
Done
lily01.rit.edu:8443

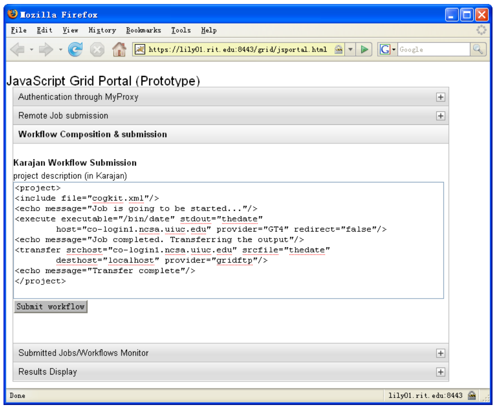
JavaScript Grid Portal (Prototype)

Remote Job submission

Job Specification

Command /bin/ls
Arguments -l
Host co-login1.ncsa.uiuc.e
Standard Output list
Provider ○GT2 ●GT4

Figure 10: Remote job definition


Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://lily01.rit.edu:8443/grid/jsportal.html
Google
Authentication through MyProxy
Remote Job submission
Submit workflow
Submitted Jobs/Workflows Monitor
Results Display
Done
lily01.rit.edu:8443

JavaScript Grid Portal (Prototype)

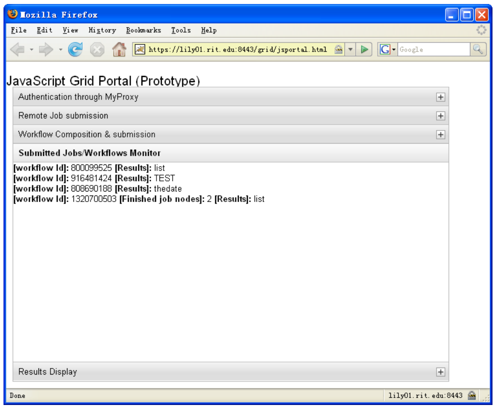
Workflow Composition & submission

Karajan Workflow Submission

project description (in Karajan)

```
<project>
<include file="cogkit.xml"/>
<echo message="Job is going to be started..."/>
<execute executable="/bin/date" stdout="thedate"
        host="co-login1.ncsa.uiuc.edu" provider="GT4" redirect="false"/>
<echo message="Job completed. Transferring the output"/>
<transfer srchost="co-login1.ncsa.uiuc.edu" srcfile="thedate"
        desthost="localhost" provider="gridftp"/>
<echo message="Transfer complete"/>
</project>
```

Figure 11: Workflow construction

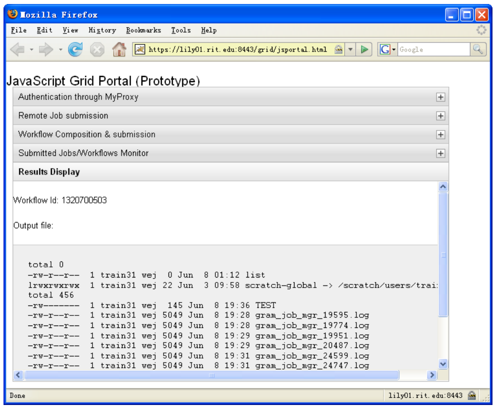

Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://lily01.rit.edu:8443/grid/jsportal.html
Google
Authentication through MyProxy
Remote Job submission
Workflow Composition & submission
Results Display
Done
lily01.rit.edu:8443

JavaScript Grid Portal (Prototype)

Submitted Jobs/Workflows Monitor

[workflow Id]: 800099525 **[Results]:** list
[workflow Id]: 916481424 **[Results]:** TEST
[workflow Id]: 808690188 **[Results]:** thedate
[workflow Id]: 1320700503 **[Finished job nodes]:** 2 **[Results]:** list

Figure 12: Job status query


Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://lily01.rit.edu:8443/grid/jsportal.html
Google
Authentication through MyProxy
Remote Job submission
Workflow Composition & submission
Submitted Jobs/Workflows Monitor
Done
lily01.rit.edu:8443

JavaScript Grid Portal (Prototype)

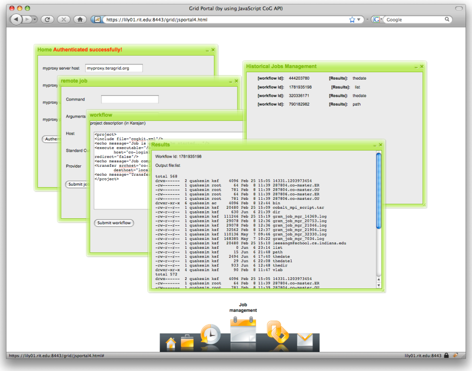
Results Display

Workflow Id: 1320700503

Output file:

```
total 0
-rw-r--r--  1 train31 wej  0 Jun  8 01:12 list
lrwxrwxrwx  1 train31 wej 22 Jun  3 09:58 scratch-global -> /scratch/users/trai
total 456
-rw-------  1 train31 wej  145 Jun  8 19:36 TEST
-rw-r--r--  1 train31 wej 5049 Jun  8 19:28 gram_job_mgr_19595.log
-rw-r--r--  1 train31 wej 5049 Jun  8 19:28 gram_job_mgr_19774.log
-rw-r--r--  1 train31 wej 5049 Jun  8 19:29 gram_job_mgr_19951.log
-rw-r--r--  1 train31 wej 5049 Jun  8 19:29 gram_job_mgr_20487.log
-rw-r--r--  1 train31 wej 5049 Jun  8 19:31 gram_job_mgr_24599.log
-rw-r--r--  1 train31 wej 5049 Jun  8 19:31 gram_job_mgr_24747.log
```

Figure 13: Result retrieval


Grid Portal (by using JavaScript CoG API)
https://lily01.rit.edu:8443/grid/jsportal4.html
Google
Home Authenticated successfully!
remote job
Command
Arguments
Host
Standard O
Provider
Submit job
Authen
workflow
Submit workflow
Historical Jobs Management
Results
Job
management
https://lily01.rit.edu:8443/grid/jsportal4.html#
lily01.rit.edu:8443

myproxy server host myproxy.teragrid.org

project description (in Karajan)

```
<project>
<include file="cogkit.xml"/>
<echo message="Job is
<execute executable="/b
        host="co-login
redirect="false"/>
<echo message="Job com
<transfer srchost="co-
        desthost="local
<echo message="Transfer
</project>
```

[workflow Id]: 444203780 [Results]: thedate

[workflow Id]: 1781935198 [Results]: list

[workflow Id]: 320336171 [Results]: thedate

[workflow Id]: 790182982 [Results]: path

Workflow Id: 1781935198

Output file:list

```
total 568
drwx------  2 quakesim ksf    4096 Feb 25 15:05 14331.1203973454
-rw-------  1 quakesim root     64 Feb  8 11:39 287804.co-master.ER
-rw-------  1 quakesim root    781 Feb  8 11:39 287804.co-master.OU
-rw-------  1 quakesim root     64 Feb  8 11:39 287806.co-master.ER
-rw-------  1 quakesim root    781 Feb  8 11:39 287806.co-master.OU
drwxr-xr-x  6 quakesim ac     4096 Feb  8 12:44 bin
-rw-r--r--  1 quakesim ksf   20480 Feb 25 15:09 cobalt_mpi_script.tar
-rw-r--r--  1 quakesim ksf     630 Jun  6 21:39 dir
-rw-r--r--  1 quakesim ksf  111246 Feb 25 15:19 gram_job_mgr_14369.log
-rw-r--r--  1 quakesim ksf   29078 Feb  8 12:36 gram_job_mgr_20753.log
-rw-r--r--  1 quakesim ksf   29078 Feb  8 12:36 gram_job_mgr_21046.log
-rw-r--r--  1 quakesim ksf   32562 Feb  8 12:37 gram_job_mgr_21904.log
-rw-r--r--  1 quakesim ksf  110136 May  7 09:46 gram_job_mgr_32330.log
-rw-r--r--  1 quakesim ksf  168385 May  7 10:22 gram_job_mgr_7034.log
-rw-r--r--  1 quakesim ksf   20480 Feb 25 15:10 leesangm@school.cs.indiana.edu
-rw-r--r--  1 quakesim ksf       0 Jun  6 23:14 list
-rw-r--r--  1 quakesim ksf      15 Jun  6 21:48 path
-rw-r--r--  1 quakesim ksf    2494 Jun  6 17:40 thedate
-rw-r--r--  1 quakesim ksf      29 Jun  6 22:08 thedate1
-rw-r--r--  1 quakesim ksf     933 Jun  6 12:48 thedir
drwxr-xr-x  4 quakesim ksf      90 Feb  8 11:47 vlab
total 572
drwx------  2 quakesim ksf    4096 Feb 25 15:05 14331.1203973454
-rw-------  1 quakesim root     64 Feb  8 11:39 287804.co-master.ER
-rw-------  1 quakesim root    781 Feb  8 11:39 287804.co-master.OU
```

Figure 14: An alternative Portal

through a component repository that we are going to be establishing.

5 Deployment

We have tested our framework on the TeraGrid as it is one of our main target platforms. Because each TeraGrid user has access to a login node, we can host the mediator on one of these nodes, as by default each TeraGrid user will have access to such a node as it is part of the user account management of TeraGrid. The result is that the client has a zero install base and the web application will provide all the essential functionalities to access Grid.

Additional JavaScript GUI components that are specifically targeting the teragrid are under development.

6 Conclusion

In this paper we presented a JavaScript-based Grid abstraction framework. It provides a client side JavaScript API, through which users can perform job and workflow management tasks on the Grid. This includes submission and monitoring of traditional jobs. However we are also working towards the development of shared workflows that are controlled not only by a single user, but by a group of users. The web services-based application backend allows that the client deployment is easy and ubiquitous due to the interoperability through WS technologies and standards. Thus, we have introduced a new and convenient tool for accessing Grids through JavaScript. Next steps include the implementation and integration of more features in regards to collaborative queue management. We also plan to investigate and support other workflows and other authentication methods.

References

[1] "Oasis soa reference model." [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

[2] "The Commodity Grid (CoG) Project." [Online]. Available: http://www.cogkit.org

[3] "The Globus Toolkit." [Online]. Available: http://www.globus.org

[4] "TeraGrid," 2001. [Online]. Available: http://www.teragrid.org/

[5] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience*, vol. 13, no. 8-9, pp. 643–662, 2001. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--cog-cpe-final.pdf

[6] S. Microsystems, "Introduction to JSR 168 - The Java Portlet Specification," Web page. [Online]. Available: http://developers.sun.com/portalserver/reference/techart/jsr168/pb_whitepaper.pdf

[7] "Open Grid Computing Environments." [Online]. Available: http://www.ogce.org

[8] J. Novotny, M. Russell, and O. Wehrens, "Gridsphere: an advanced portal framework," in *Euromicro Conference, 2004. Proceedings. 30th*, 2004, pp. 412–419.

[9] "TeraGrid Portal." [Online]. Available: http://www.teragrid.org/userinfo/portal.php

[10] "Same Origin Policy for JavaScript." [Online]. Available: http://developer.mozilla.org/En/Same_origin_policy_for_JavaScript

[11] IETF, "HTTP State Management Mechanism," Feb 1997. [Online]. Available: http://www.w3.org/Protocols/rfc2109/rfc2109

[12] G. Di Lucca, A. Fasolino, M. Mastoianni, and P. Tramontana, "Identifying cross site scripting vulnerabilities in Web applications," in *Web Site Evolution, 2004. WSE 2004. Proceedings. Sixth IEEE International Workshop on*, 11 Sept. 2004, pp. 71–80.

[13] J. Shanmugam and M. Ponnavaikko, "A solution to block Cross Site Scripting Vulnerabilities based on Service Oriented Architecture," in *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*, 11-13 July 2007, pp. 861–866.

[14] N. Jovanovic, E. Kirda, and C. Kruegel, "Preventing Cross Site Request Forgery Attacks," in *Securecomm and Workshops, 2006*, Aug. 28 2006-Sept. 1 2006, pp. 1–10.

[15] O. Ismail, M. Etoh, Y. Kadobayashi, and S. Yamaguchi, "A Proposal and Implementation of Automatic Detection/Collection System for Cross-site Scripting Vulnerability," in *Proc. 18th International Conference on Advanced Information Networking and Applications AINA 2004*, vol. 1, 2004, pp. 145–151 Vol.1.

[16] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: a Client-side Solution for Mitigating Cross-site Scripting Attacks," in *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2006, pp. 330–337.

[17] "The TLS Protocol Ver 1.0." [Online]. Available: http://tools.ietf.org/html/rfc2246

[18] OASIS, "Web Services Security v1.0 (WS-Security 2004)," 2004. [Online]. Available: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[19] "Grid Security Infrastructure." [Online]. Available: http://www.globus.org/security/

[20] G. von Laszewski, A. J. Younge, X. He, and F. Wang, "Gridshell: Interactive task management for grid and cluster computing," (submitted for review), Sep. 2008.

[21] Y.-C. Lee, C.-M. Ma, and S.-C. Chou, "A service-oriented architecture for design and development of middleware," in *Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific*, 15-17 Dec. 2005, p. 5pp.

[22] A. Uyar, W. Wu, H. Bulut, and G. Fox, "Service-oriented architecture for a scalable videoconferencing system," in *Pervasive Services, 2005. ICPS '05. Proceedings. International Conference on*, 11-14 July 2005, pp. 445–448.

[23] X. Lu, "An investigation on service-oriented architecture for constructing distributed web gis application," in *Services Computing, 2005 IEEE International Conference on*, vol. 1, 11-15 July 2005, pp. 191–197vol.1.

[24] ECMA, "Standard ecma-262 ECMAscript language specification, 3rd edition," Dec 1999. [Online]. Available: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[25] K. Amin, M. Hategan, G. von Laszewski, and N. J. Zaluzec, "Abstracting the Grid," in *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2004)*, La Coruña, Spain, 11-13 Feb. 2004, pp. 250–257. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--abstracting.pdf

[26] G. von Laszewski, "Java CoG Kit Workflow Concepts," *Journal of Grid Computing*, Jan. 2006, http://dx.doi.org/10.1007/s10723-005-9013-5. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-workflow-taylor-anl.pdf

[27] G. von Laszewski, T. Trieu, P. Zimny, and D. Angulo, "The Java CoG Kit Experiment Manager," Argonne National Laboratory, Tech. Rep., Jun. 2005. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-exp.pdf

[28] IETF, "The application/json Media Type for JavaScript Object Notation (JSON)," Jul 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4627.txt

[29] JSON, "Introducing JSON." [Online]. Available: http://json.org/

[30] "The Dojo Toolkit." [Online]. Available: http://dojotoolkit.org/

[31] J. Walker, "JSON is not as safe as people think it is," March 2007. [Online]. Available: http://directwebremoting.org/blog/joe/2007/03/05/json_is_not_as_safe_as_people_think_it_is.html

[32] J. Grossman, "Advanced Web Attack Techniques using GMail," Jan 2006. [Online]. Available: http://jeremiahgrossman.blogspot.com/2006/01/advanced-web-attack-techniques-using.html

[33] R. Yates, "Safe JSON," March 2007. [Online]. Available: http://robubu.com/?p=24

[34] W3C, "Simple Object Access Protocol (SOAP) version 1.1," May 2000. [Online]. Available: http://www.w3.org/TR/soap/

[35] S. Microsystems, "Java API for XML Web Services (JAX-WS)." [Online]. Available: https://jax-ws.dev.java.net/

[36] D. Jayasinghe, "Axis2 Deployment Model." [Online]. Available: http://jaxmag.com/itr/online_artikel/psecom,id,757,nodeid,147.html

[37] Apache, "Apache Axis2." [Online]. Available: http://ws.apache.org/axis2/index.html

[38] ——, "Apache Tmocat." [Online]. Available: http://tomcat.apache.org/

[39] "Maven." [Online]. Available: http://maven.apache.org/

[40] NCSA, "MyProxyLogon." [Online]. Available: http://grid.ncsa.uiuc.edu/myproxy/MyProxyLogon/

[41] Apache, "Apache CXF Framework." [Online]. Available: http://cxf.apache.org/

[42] "Shiboleth." [Online]. Available: http://shibboleth.internet2.edu/