JavaScript Grid Abstractions

Gregor von Laszewski[1], Fugang Wang[1], Andrew Younge[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 Grid abstraction framework that allows access to the Grid infrastructure using JavaScript while leveraging the power of current Grid middleware and upperware toolkits such as the Globus Toolkit and the Java Commodity Grid (CoG) Kit . The system is heavily based on Web 2.0 technologies and allows accessing the Grid through a Service-Oriented Architecture. An application interface in JavaScript is provided to enable developers to access Grid services from JavaScript. Moreover, our framework includes additional services to enable the creation of advanced Grid services. The availability of our framework simplifies not only the development of new services but also the development of advanced client side Grid applications. We demonstrate this ability while providing a mechanism to develop Grid workflows through advanced services and a graphical user interface defined in JavaScript. Overall, Grid developers will have another tool at their disposal that projects a simpler way to distribute and maintain software while at the same time being able to deliver quickly advanced interfaces and social services for the scientific community.

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

1 Introduction

Scientific users has been leveraging the power of Grid Computing for years, as it provides computing power unlike anything else. The heart of Grid computing consists of middleware which unites many administratively and spatially separate resources under one Grid system. One such example of this middleware is the Globus Toolkit [1], which introduces a number of complexities to accomplish its task. Often these complexities are passed directly to the user community.

For many scientific users, the challenges introduced by using Grid computing often are too great to justify using Grids at all. Thus, there is a need to provide a simplified user environment without compromising the power of the Grid system. The Java CoG Kit [2] aims to do the same by providing a simple interface where scientific users can effectively utilize Grid technologies without all of the challenges that typically come with it.

In this paper we propose to provide a JavaScript Grid abstraction framework. It lies on top of the existing Grid middleware such as the Java CoG Kit and provides a Graphical User Interface (GUI) in JavaScript. Various Grid services are exposed through the Web Client side JavaScript API, while the Mediator interacts with the underlying grid services and deploys them as Web services. Using the JavaScript API, Grid users receive a more convenient way to access Grid services. The Grid developers can also benefit from this framework as they can develop new Grid services and applications more efficiently.

2 Architecture

The framework can be divided into three logical components, illustrated in Figure 1. The first component is the Web Client where we provide GUI portal to Grid user. This encompasses functionality such as workflow browsing, editing, job submittion and execution status querying. The interaction with the server side is carried on through a JavaScript API. The second part is Mediator, which communicates with Grid services through underlying Grid middleware such as the Globus Toolkit using the Java CoG Kit. Meanwhile the mediator exposes the services through Web Services to the Web Client. It also introduces a CoGQueue management mechanism, through which Grid users can share their workflows and collaborate together. The third part consists of Support Services. It contains constructs like user account management, and job execution status querying. These support services extend beyond the normal Globus functionality to provide additional features that are otherwise not possible.

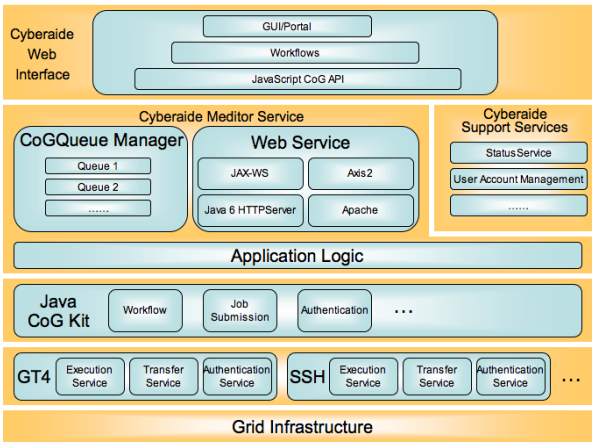

Figure 1. The System Architecture

3 Overview

This section introduces the purpose of each component of the system.

3.1 Web Client

The Web client provides a dynamic Graphical User Interface (GUI) in the form of a Grid Portal. Within the portal, high-level Grid functionality is exposed. This is accomplished through the use of Javascript, HTML, XML and CSS to provide a seemless environment for users.

On the web client, the user can perform basic tasks common on most Grid systems. This includes creating and editing new tasks and workflows, submitting the tasks and workflows to the grid system automatically, queueing tasks to a collaborative queue, and monitoring tasks as they progress through the Grid environment.

3.2 Mediator

The Mediator acts as a translator that exists between the web client and the underlying Grid services. Requests from the Web client come into the Mediator and are interpreted into commands that initialize the desired Grid functionality. The Mediator service is a crucial part of the system as it is the point at which abstract actions from the user are turned into complicated interactions with the underlying Grid infrastructure.

The Mediator uses web services to provide functionality calls that communicate with the Web client. The connection should be secured through HTTP over SSL/TLS (HTTPS). Message level security based on the WS-Security specification [3] could also be integrated to enforce the endpoint-to-endpoint security. The web services can be developed and deployed by using JAX-WS [4] from Java SE 6 or Axis2 [5] from Apache. The mediator functionality utilizes the Java CoG Kit which in turn interacts with the underlying Grid infrastructure.

The Mediator service can support proxy credential delegation so users won't have to authenticate more than once, effectively implementing a single-sign-on mechanism. It can be divided into two physical parts to deploy to gain an improved security. In such case, the two parts can communicate with each other using Web services or a Remote Method Invocation(RMI) approach. The first part wroks as a web service agent, and the call to web services will be forwarded to the othe part, which could be behind a firewall.

3.3 Support Services

The support services component provides valuable tools and services to aid the mediator and Web client when working in both a single user and collaborative environment. These services includes include account creation and management, collaborative queueing and resource sharing, persistent storage of task execution states, and integrated Web 2.0 mashup technologies. These services offer the user a rich experience that extend the usefulness of current Grid infrastructure in a way that is easy and convenient for the user.

4 Design Issues

The system design is based heavily on Object-oriented programming model and Service-Oriented Architecture (SOA) [6,7]. The entities in the system are all objects, while Service-oriented architecture is used for the interaction between the distributed objects. The benefit of this approach is that it makes the distributed objects loosely coupled, so we are concerned only with the contract between them, not the implementation. As long as we keep the contract between entities, changes on implementation of one part would not affect other parts of the system.

4.1 JavaScript CoG Client Functionality

In order to interact with Grid, we need to deal with job/workflow submission. We also need to deal with workflow queue, authentication, and status query. This section summaries the design of JavaScript library to facilitate that.

4.1.1 Job Management

To deal with job management, we need a JavaScript object which is basically an abstraction for all executable entities, jobs or workflows.

It contains two important fields, *Attributes* and *Provider*. Provider could be *system*, *Karajan* or another workflow engine's name. Attributes would be commands or the content of the workflow description while using workflow stated in Provider.

Currently we are interested with the *Karajan* engine since it is the default workflow description language used in Java CoG Kit. We could have class KarajanWorkFlow to provide the workflow browsing and editing functionality in the web client side.

4.1.2 Workflow Management

Workflow is a set of jobs with dependencies between each other. As stated previously, we are currently have special interests with *Karajan* workflow. Therefore we need an object to depict the *Karajan* workflow.

A Karajan workflow can contain one or more jobs. It supports hierarchical workflow, which means a workflow can be a job in another workflow.

It provides the following functionality:

- addJob (jobname, job) – A new job is appended to a workflow.
- deleteJob (jobname) – The job with the name specified by parameter jobname is deleted. As a result, all related dependency is deleted as well.
- listJobs( ) – List all jobs in a workflow.
- searchByName(jobname) – Search job by name in a workflow
- addDependency (jobparent, jobchild) – Add dependency which says that job jobparent should be executed before job jobchild.
- removeDependency(jobparent, jobchild) – Remove dependency between job jobparent and job jobchild.
- toggleDependency(jobparent, jobchild) – If there is a dependency, remove it. If there is not, create a new one.

Support for other workflow formats will need some other similar class depending on the workflow engine used.

4.1.3 Queue Management

To deal with job/workflow queue, we need a JavaScript object to represent a queue of *Executable* objects that have no dependency on each other. In other words, they can be executed in arbitrary order. Notice that in the following context, we use the name workflow to indicate all the kinds of Executable objects, whether theyre instance of some workflows like *Karajan* or an executable scripts.

Queue management includes functionality to enable client-sde workflow queue management:

- addWorkflow(name, workflow)– Append the workflow specified by parameter to a workflow queue.
- removeWorkflowByIndex(index) – Remove a workflow according to parameter index.
- removeWorkflowByName(name) – Remove a workflow with name specified by parameter name.
- clearAll( ) – Remove all workflows in the queue.
- searchByName(name) – Return the index of the workflow with the name specified by parameter. If no corresponding workflow exists, return -1.

4.1.4 Authentication

The Authenticator is an abstraction method that is used to handle authentication. It the same two fields of *Attributes*, and *Provider*, and an *authenticate()* function. We use the default HTTPS connection as provider, and add support for other authentication methods. The Attributes contain the necessary information to authenticate the user while use the method provided by the Provider. For the case of HTTPS, it would contain the user's username and password.

4.1.5 Workflow Status

To monitor the execution status, we provide a Query service that is used to query state of workflows that the user submitted to execute.

It provides the following functionality:

- query() – Get state of all workflows submitted by the user.
- query(workflowIDs) – Get state of specified workflows submitted by the user.

4.1.6 JavaScript CoG Abstractions

Now we have the above JavaScript objects defined, which function as classes in a Object Oriented (OO) design. By using these classes, we define utility class CoGClass to do the workflow submission and result query.

The functionality provided is listed below:

- authenticate(Authenticator) – The Authenticators authenticate() function will be called. For example, if we use HTTPS as provider, then a username/password authentication will be carried out.

- execute(Executable, resources) – Submit a workflow to server side to execute. The resources specify necessary resources associated with the jobs. transfer(from, to); transfer data from from to to.
- query() – Get state of all workflows that were submitted to execute by the user.
- query(workflowids) – Get state of specified workflows that were submitted to execute by the user.

CoGClass will perform the interactions with the Grid through server side web services. Additionally, an elementary collaborative environment is provided, in a form of shared workflow queue. Class CoGQueue deals with the interactions between the client and the server sides CoGQueue zone.

It provides the following functionality:

- listQueues() – List all the queues that the user are participating.
- grantAccess(queueID, userlist) – The owner of a queue can give some other users access to the queue.
- add(queuename, Executable) – Submit a workflow into the server sides CoGQueue zone to store for future use or share with other users. If the queuename exists and the user has privilege to access that queue(he is a participants of that queue), the server side program will extract info from the Executable object and construct objects to insert into the queue; otherwise the server side program will construct a new queue with that name, and set the user as the owner of the queue.
- remove(queueID, sharedWorkFlowID) – The owner of a workflow can remove it.
- list(queueID); list all workflows metadata shared in the queue.
- listParticipants(queueID) – List all the participants of the queue.
- listByName(queueID, username) – List the workflow's metadata owned by a user from one queue.
- listByType(queueID, provider) – List all the workflow's metadata with the specified type, say, Karajan workflow, from the specified queue.
- getStatus(queueID, sharedWorkFlowID) – Query the specified workflow's status. It could be edited by another user.
- browseWorkflow(queueID, sharedWorkFlowID) – Download the workflow to client side to browse.
- obtainWriteToken(queueID, sharedWorkFlowID) – The user try to obtain the write token.
- editWorkflow(queueID, sharedWorkFlowID) – The user will try to obtain the write token and then to edit the workflow.
- updateWorkflow(queueID, sharedWorkFlowID) – Update the workflow when complete editing.

Through the CoGQueue, a user could fulfill workflow persistent before submission, and share workflows among a certain group. This enables collaboration among Grid users.

4.1.7 Security concerns

Since the HTTP protocol is stateless, we may need to maintain user's status in client side during one session. The status is kept in a cookie file, which may include user related information and/or session specified information. The paradigm here is that we need to keep as few sensitive data in client side as we can, since the client side is always the weakest one in the security chain. It may be controlled and abused by malicious users. Also it's vulnerable to some JavaScript related attacks.

The JavaScript security issue has two kinds of meanings. From the client side's point of view, the script from the server side should not harm the client side; while from the server's side, we cannot assume the data from client side are sent by a trusted, secured client.

Two policies are enforced by most browsers to deal with this issue. The first is sandbox, which means the script files from the server can only be executed in a limited manner in a restricted environment which has no impact in the client system; the second is same-origin policy, which assures that script from a certain server can only access resources from the same origin. Cross-site Scripting(XSS) [8] is a typical attack which violates the same-origin policy. By deliberately coding, a malicious attacker may forge a legal request which the browser assumes it originates from the same place as one of the running scripts, thus expose the user's sensitive information associated with the web site where the running script is from. Or the malicious code may forge the user's request to the website, which is called Cross-site Request Forge(XSRF) [9], thus performing operations that the user doesn't intent to do. There is no single method to deal with these attacks, while set of rules need to be considered during the development of the JavaScript code [9, 10].

4.2 Mediator Service

4.2.1 Web Services

We use Java to develop and deploy the web services. We can implement the web Service Endpoint Implementation(SEI)

class by using annotation mechanism provided by Java SE 6. JAX-WS provides a convenient tool wsgen [11] that can generate all the artifacts required for web service deployment from a Java SEI class. An alternative method to accomplish this is using axis2 related tools. To deploy the web service in some web container like Apache/Tomcat, or Java SE 6s built-in Httpserver will expose the web service to the client side.

Security is obtained through a HTTPS connection and message level security based on the WS-Security policy. Users need to authenticate to the server using a username and password. Furthermore, all the messages come from the client side need to be semantically checked in case of the request is coming from altered malicious script code and being used to attack the Grid system.

4.2.2 Application Logic layer

The application logic layer deals with Grid service interations. It utilizes Java CoG Kit API to interact with the underlying Grid infrastructure. The call to the Java CoG Kit API will be transformed into a Web service and be exposed to the JavaScript CoG API.

The proxy credential delegation happens here. The JGlobus module from Java CoG Kit provides functionalities to handle the interaction with MyProxy. The MyProxy provides MyProxyLogon API [12] which can also fulfill this task. As long as a user stored his/her credential on a MyProxy server, the user can retrieve the credential by provide the username and passphrase used when generating the proxy credential. Thus the user delegates the application server to communicate with the underlying Grid infrastructure during the user session, without the need to authenticate himself/herself each time when submitting a job or query status of a submitted job.

4.2.3 CoGQueue Management

This is the space where users share their workflows and do collaborative tasks. The space contains queues of sharedWorkFlow. We could also implement this as a generic object queue.

A queue is defined as:

$$queue \quad ::= \quad (\textbf{\textit{queueID}}, label, owner, participants, objects)$$

- *QueueUID* : Unique ID in this CoGQueue zone;
- *owner*: Who created this queue. Only the owner can grant other users access to the queue.
- *participantsList*: Who are participating in this queue. Could be regarded as a Access Control List of this queue.
- *objects*: Those shared workflows.

Objects in the queue, or called SharedWorkFlow, are defined as:

$$workflow \quad ::= \quad (\textbf{\textit{SWFUID}}, label, owner, status, writeToken, type, \textbf{\textit{WFObj}}),$$

- *SWFUID*: The unique id for the shared workflow
- *owner*: The workflow's owner. It's the person who uploaded the workflow.
- *status*: Indicate whether or not this workflow is being edited by a user.
- *writeToken*: Works as a lock, to guarantee atomic write operation of the workflow object, thus ensuring the content is consistent. The server needs to check the client's liveness periodically to make sure it's there editing the workflow. Otherwise the writeToken will be recovered thus make the workflow available for other users to edit.
- *type*: The object's type. Could be system executable script or Karajan workflow for example.

We could also maintain a user/SharedWorkFlow map to speed up some operations called from client side.

4.3 Implementation

We have developed a prototype of the system, which can handle job submission and status query from web client side; provides a GUI based environment in the Web client side to mange and edit the jobs/workflows. The prototype utilizes Java CoG Kit as intermediary services with Axis2 and Tomcat to develop and deploy web services. The Karajan workflow engine is supported on the client side and is used to composite workflows.

4.3.1 Job construction and workflow composition

The prototype now supports system commands and Karajan workflow engine based job/workflow. This is determined by whether or not the provider is System or Karajan. The GUI widget provides easy way to define a job through the Karajan workflow language.

A user only need to select elements from panel, then the workflow description will be generated automatically.

Grid users can also construct a more complicated workflow, by adding dependency between jobs. In this logical level, a workflow is a Directed Acyclic Graph (DAG) where arrows between jobs indicate job dependency.

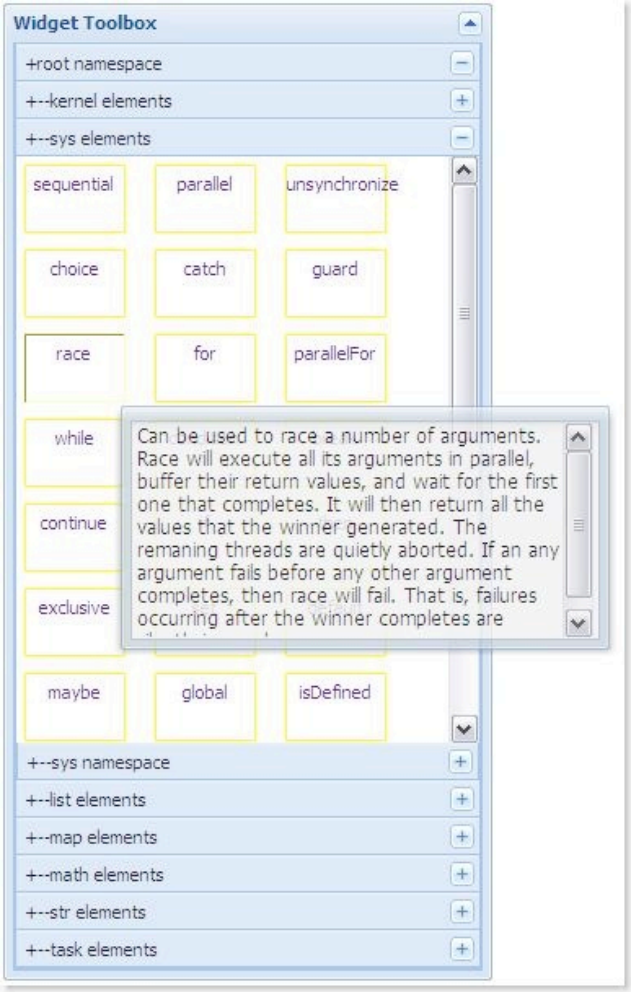

Figure 2. Job definition

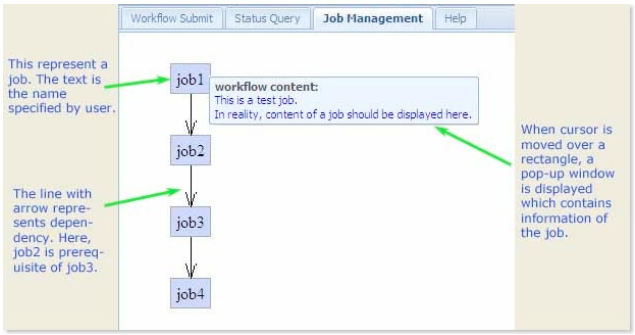

Figure 3. Workflow construction

4.3.2 Job execution

When finished job construction, it is then ready to submit for execution. The web client will invoke server side web service, and the server side will submit the job by utilizing the related API from Java CoG Kit. Then a jobId will be returned to the client side as a handler. The user can query job execution status by using this id. The response from the server will be displayed in the panel.

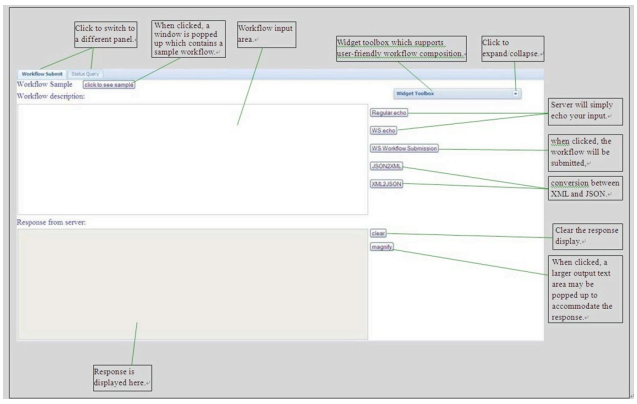

Figure 4. Job submission

4.3.3 Status query

Users can check state of the submitted workflows by specifying the workflow id, or get states of all the submitted workflows. The response from server will be displayed in the panel.

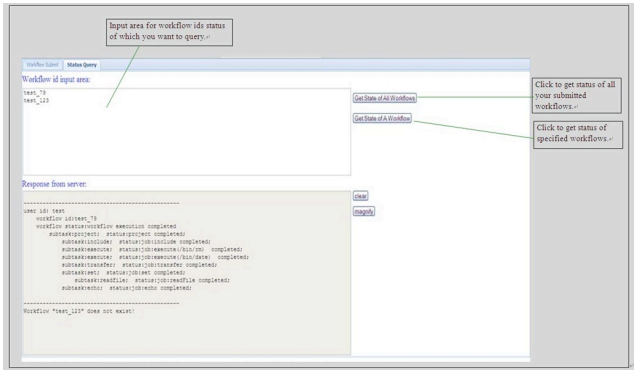

Figure 5. Job status query

5 Conlcusion

In this paper we present a JavaScript Grid abstraction framework. It provides a client side web application through which Grid users can deal with job and workflow management in client side, share workflows with other Grid users during collaboration, submit jobs to Grid services to execute, and query status of the jobs. The web services based application takes advantages of easy deployment and

great interoperability. This introduces a more convenient and user-friendly Grid computing environment. We have developed a prototype which can carry on the basic Grid operations. The next step is implementing and integrating CoGQueue, the user collaboration part. We also plan to investigate and support other workflows and other kinds of authentication methods.

Acknowledgment

Work conducted by Gregor von Laszewski is supported (in part) by NSF CMMI 0540076 and NSF SDCI NMI 0721656.

Status

This paper has been enhanced from its originial form from 2007. An updated implementation of this paper has since been published [13].

References

[1] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997, ftp://ftp.globus.org/pub/globus/papers/globus.pdf.

[2] G. von Laszewski, "A Loosely Coupled Metacomputer: Cooperating Job Submissions Across Multiple Supercomputing Sites," *Concurrency, Experience, and Practice*, vol. 11, no. 5, pp. 933–948, Dec. 1999, the initial version of this paper was available in 1996. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--CooperatingJobs.pdf

[3] 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

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] "JAX-WS wsgen Document." [Online]. Available: http://java.sun.com/javase/6/docs/technotes/tools/share/wsgen.html

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

[13] G. von Laszewski, F. Wang, A. Younge, X. He, Z. Guo, and M. Pierce, "Cyberaide javascript: A javascript commodity grid kit," in *GCE08 at SC08*. Austin, TX: IEEE, Nov. 16 2008. [Online]. Available: http://cyberaide.googlecode.com/svn/trunk/papers/08-javascript/vonLaszewski-08-javascript.pdf