Providing Dynamic Virtualized Access to Grid Resources via the Web 2.0 Paradigm

Luca Clementi
National Biomedical Computation Resource
Center for Research in Biological Systems
University of California, San Diego
9500 Gilman Dr.
La Jolla, CA
+1-858-534-6779
lclement@ucsd.edu

Zhaohui Ding
College of Computer Science and Technology, Jilin University
2699 Qianjin Street
Changchun,130012 China
+86-431-85166885
zhaohui.ding@email.jlu.edu.cn

Sriram Krishnan
National Biomedical Computation Resource
San Diego Supercomputer Center
University of California, San Diego
9500 Gilman Dr.
La Jolla, CA
+1-858-822-5425
sriram@sdsc.edu

Xiaohui Wei
College of Computer Science and Technology, Jilin University
2699 Qianjin Street
Changchun,130012 China
+86-431-85166885
weixh@jlu.edu.cn

Peter W. Arzberger
National Biomedical Computation Resource
Center for Research in Biological Systems
University of California, San Diego
9500 Gilman Dr.
La Jolla, CA
+1-858-534-5079
parzberger@ucsd.edu

Wilfred Li
National Biomedical Computation Resource
San Diego Supercomputer Center
University of California, San Diego
9500 Gilman Dr.
La Jolla, CA
+1-858-822-0974
wilfred@sdsc.edu

ABSTRACT

Grid systems provide mechanisms for single sign-on, and uniform APIs for job submission and data transfer, in order to allow the coupling of distributed resources in a seamless manner. However, new users face a daunting barrier of entry due to the high cost of deployment and maintenance. They are often required to learn complex concepts relative to grid infrastructures (credential management, scheduling systems, data staging, etc). To most scientific users, running their applications with minimal changes and yet getting results faster is highly desirable, without having to know much about how the resources are used. Hence, a higher level of abstraction must be provided for the underlying infrastructure to be used effectively. For this purpose, as part of our prior work, we have developed the Opal toolkit for exposing applications on grid resources as simple Web services [1]. Opal provides a basic set of APIs that allows users to execute their deployed applications through graphical user interfaces, or via programmatic means.

In this paper, we present our ongoing work to extend the Opal toolkit that enables the creation of an end-to-end infrastructure to dynamically leverage scientific applications on grid resources. In particular, we focus on the following two key extensions. We have developed a command-line syntax description language that can be published via the Opal interface. This enables the creation of dynamic Web forms for job submission, and may even be used for creating familiar application-specific user interfaces in other types of Problem Solving Environments (PSE). Additionally, we have extended the Community Scheduler Framework (CSF4) Meta-scheduler to support the concept of applications as first class resources. This enables Opal to submit jobs to different grid resources where an application is deployed without affecting how user applications interact with Opal. We believe that these extensions enable the Opal toolkit to effectively leverage multiple grid resources, and provide access to the users in a transparent Web 2.0 fashion.

Categories and Subject Descriptors

D.4.7 [**Organization and Design**]: Distributed systems – *Design.*

Keywords

Grid, Service Oriented Architecture, Web 2.0.

1. INTRODUCTION

Grid systems have become very popular in recent times for coupling geographical distributed resources. However, it is common knowledge that they are quite complicated to deploy and maintain for system administrators and middleware developers. Furthermore, end-users are typically burdened with learning complicated concepts such as credential management, scheduler access, data transfer protocols, etc., in order to exploit the capabilities provided by grid systems. We believe that significant improvements can be made in making grid resources easier to use – firstly, for developers and system administrators to deploy and develop grid applications, and secondly, for scientific end-users to use.

The Web 2.0 paradigm provides several lessons that can be leveraged by the scientific and grid communities. In "What is Web 2.0" [WEB2.0], O'Reilly lists some design patterns and general recommendations that are a large part of the Web 2.0 paradigm. In particular, he stresses that developers should (1) use the Web as a platform for development, (2) harness the collective intelligence of the community, (3) focus on services, and not pre-packaged software, with cost-effective scalability, (4) enable light-weight programming models, and (5) provide a rich user

experience. In a recent paper [17], we list our positions on Services Oriented Architectures (SOA) for e-Science. To summarize, scientific SOA developers should (1) focus on application-level services, (2) provide access to virtualized resources, (3) enable access via multiple user interfaces, and (4) leverage the right tools (in our case, open source commodity based software). With the advent of the Web Services Resource Framework (WSRF) [8], the grid community has been progressing towards some of these principles – especially in using service orientation and enabling access to tools via Web service protocols. However, to close the loop, we believe that there needs to be an effort to provide access to these scientific applications via user interfaces that are dynamic, and easy to deploy and maintain (similar to the dynamic interfaces to Web tools provided by mechanisms like AJAX).

In this paper, we present our ongoing development efforts to provide useful extensions to both Opal [1] and the CSF4 Meta-scheduler [7]. In particular, we have developed a new Web based utility for the Opal toolkit which is able to dynamically generate submission forms based on command line arguments, and a new extension to the CSF4 meta-scheduler which allows scheduling based on applications as first class resources. Finally, we discuss how these two applications are integrated together to achieve a simpler deployment process, along with the provision of high-level graphical user interfaces.

The rest of the paper is organized as follows. In Section 2, we present an introduction to the Opal toolkit, on which the new work is based on. In Section 3, we describe our approach to automatic interface generation. In Section 4, we discuss how we expose applications as resources via the CSF4 Meta-scheduler framework. In Section 5, we tie all the pieces together into an end-to-end system. We discuss some related work in Section 6, and present our conclusions in Section 7.

2. BACKGROUND

One of the goals of the National Biomedical Computation Resource (NBCR) is to provide transparent access to grid resources by "grid-enabling" scientific codes, in order to enable novel scientific workflows for multi-scale biomedical applications [14]. As part of our previous work, we have developed the Opal toolkit [1] for wrapping scientific applications as Web services. In order to wrap an application as a Web service, application developers are expected to write a simple XML-based application configuration. The application configuration contains information about the scientific application, such as application name, binary location, and application metadata (e.g. usage information). Using a simple Apache Ant task, the Opal toolkit can deploy the application as a Web service into a container based on Apache Tomcat and Axis. Once the application is deployed as a Web service, clients can access this service programmatically using its WSDL description. The WSDL API provides operations for job launch (which accepts command-line arguments and input files as its parameters), querying status, and retrieving outputs. Furthermore, it also provides an API for retrieving application metadata. WSDL savvy users could write their own clients to access Opal based applications. However, not every scientific user is capable or interested in writing Web service clients. Hence, several interfaces have been provided for the end-user. Apart from command-line clients written in languages like Java, Python [4] and Perl [15] available from NBCR, Web interfaces for Opal based services are available through the GridSphere portal environment [18], such as the MEME portlet in My WorkSphere [24]. In addition, Opal based interfaces via Rich Internet Applications (RIA) such as Gemstone [20], and Problem Solving Environments (PSE) such as the Python Molecular Viewer (PMV) [22], AutoDockTools [21] and Continuity [27] have also been provided.

3. INTERFACE DEPLOYMENT

As the number of services deployed using Opal increases, it becomes impractical to provide user interfaces for every service on every Web, PSE or RIA platform. While one could write an interface for a service on every platform, it is clearly not very scalable from a software engineering standpoint. We believe that it is a better idea to have the service describing its usage in such as way that it can be translated into user interfaces for different platforms. For this reason, we have extended Opal to enable application developers to describe the command-line syntax of their applications. Using this syntax, we have created a Web interface that can dynamically create submission forms based on the input and output of applications.

3.1 Opal extension

To improve the Opal deployment process we added an optional element to its XML deployment configuration file. In our experience with working with a number of scientific applications, we noticed that the command line arguments for most scientific applications consist of the following members:

- Flags: These parameters are not ordered and are usually represented with a character prefixed with a dash and they activate a functionality in the application (-verbose)
- Tagged Parameters: These parameters are usually formed by a prefix and input word (-input <filename>), they can appear in any order
- Untagged parameter: These parameters are not prefixed therefore their order is relevant

Hence, we added an optional element to the XML configuration for a scientific application, called *args*, which contains the above elements. Each of the above elements also consists of other meta-data, e.g. the type for tagged or untagged parameters such as Integer, Float, Boolean, the default values, a short text description, etc. Moreover we provide the possibility to group parameters that supply similar functionalities. Eventually a group can also be defined as exclusive, which implies that only one parameter can be selected among the ones belonging to that group. The following XML snippet shows an example of a configuration file for the PDB2PQR application, which provides a routine to translate between the PDB and PQR molecule formats [4].

```
<args xmlns="http://nbcr.sdsc.edu/opal/types">
  <!-- list of flags -->
  <flags>
```

Luca Clementi, Zhaohui Ding, Sriram Krishnan, Xiaohui Wei, Peter W. Arzberger, Wilfred Li, Providing Dynamic Virtualized Access to Grid Resources via the Web 2.0 Paradigm, International Workshop on Grid Computing Environments 2007, Nov., Day, 2007, Reno, NV, USA. An electronic version of this document will be available at http://casci.rit.edu/proceedings/gce2007

```
    <flag>
      <id>nodebump</id>
      <tag>--nodebump</tag>
      <textDesc>
        Do not perform the debumping operation
      </textDesc>
    </flag>
    …
  </flags>

  <!-- list of tagged parameters -->
  <taggedParams>
    <separator>=</separator>
    <param>
      <id>ffout</id>
      <tag>--ffout</tag>
      <paramType>STRING</paramType>
      <textDesc>Instead of using the standard
        canonical naming scheme for residue and
        atom names, use the names from the given
        forcefield</textDesc>
    </param>
    …
  </taggedParams>

  <!-- list of untagged parameters, in order -->
  <untaggedParams>
    <param>
      <id>output-path</id>
      <paramType>FILE</paramType>
      <ioType>OUTPUT</ioType>
      <textDesc>The desired output name of the PQR
        file to be generated</textDesc>
    </param>
    …
  </untaggedParams>

  <groups>
    <group>
      <name>inputParam</name>
      <elements>inFile inId</elements>
      <required>true</required>
      <exclusive>true</exclusive>
      <textDesc>Input file to be used (choose
      one of the two options)</textDesc>
    </group>
    …
  </groups>
</args>
```

While this model is fairly simplistic, it is capable of describing the syntax of most command-line arguments. In its current form, it is not possible to describe semantic information of the various types, or relations between the values of various parameters. This will be addressed as part of our future work.

3.2 Automatic Interface Generator

We follow a two-fold approach to interface generation. For users who don't wish to specify an argument specification, a generic form is provided that provides users with fields for selecting input files and typing command-line arguments. While this enables basic job submission and status queries, it does not quite fit the rich user environment model promoted by the Web 2.0 paradigm. Hence, we also provide a more advanced submission interface that can fetch the deployment configuration from the Opal service via its WSDL API at run time, and create a personalized input form based on the *args* element. The form is created following these simple rules:

- Flag parameters are rendered as a check boxes
- Tagged and Untagged parameters are rendered as:
  - Check boxes if their type is Boolean
  - File input fields if their type is file and they are input parameters
  - Radio buttons if their type is a list of exclusive values
  - Input text fields for all the other cases

All the groups are displayed using a descriptive heading and then listing the associated parameters. If the group is exclusive every listed parameter is associated with a radio button which can activate the selected input field and deactivated the others. Opal GUI can also query Opal server and dynamically display a list of available services.

Figure 1 shows a typical scenario where a user requests the submission form for the PDB2PQR service. The Opal GUI dynamically queries the Opal server to fetch the deployment configuration file, and a customized submission form is rendered to the user. Once the submission button is pushed, the Opal GUI launches the PDB2PQR via the Opal Web services API.

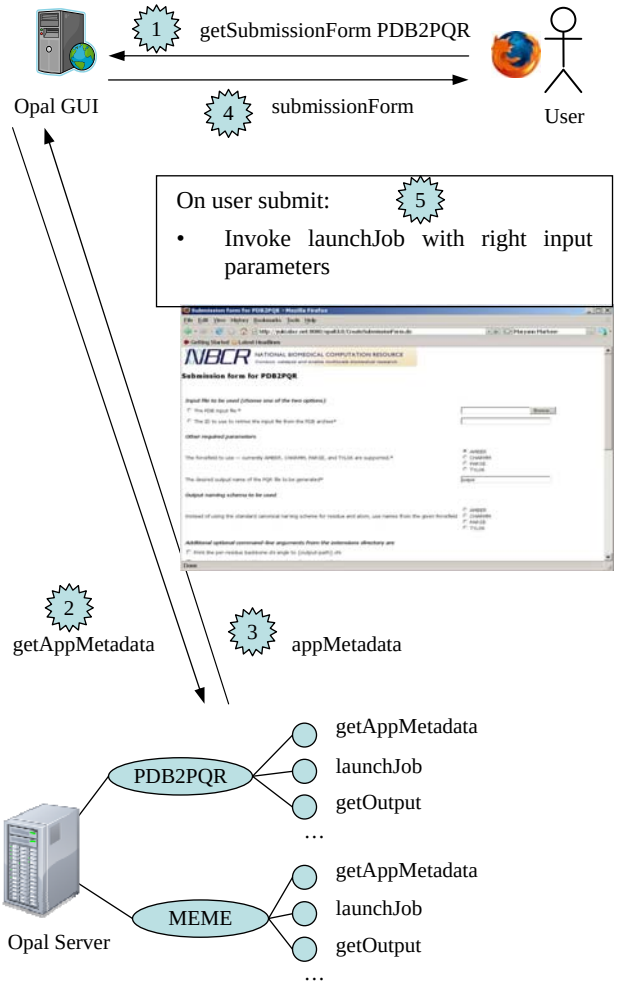

Figure 1 - Service invocation with Opal GUI

While we demonstrate the use of this approach for Web interfaces, it is general enough to be used by any environment. In short, a PSE that wishes to dynamically generate Opal interfaces must provide an Opal shell that can request the application configuration at run-time, and translate it to the presentation logic of the corresponding system. This is possible because, unlike other approaches like WSRP [29], the Opal services do not push any presentation logic. They only enable access to application metadata in XML form, which is easy to translate into a number of markup languages (even as trivially as via XSLT transformations).

Currently our user interface does not provide security mechanism but we plan to support HTTPS between the client browser and the Opal GUI server, with username and password based authentication. The Opal GUI can rely on third party security systems in order to obtain centralized users credentials (e.g. GAMA [23]). Then Opal GUI can adopt GSI [16] based HTTPS for the job submission to the remote back-end Opal services.

4. Virtualized Resources

The Opal toolkit models applications as resources while exposing them as Web services; however, the base version can only submit jobs to a local scheduler. For greater scalability, it is necessary to leverage multiple clusters located at different locations. However, our goal is to do this without compromising the ease of use of application deployment via Opal. With that in mind, we extend the concept of an application as a requestable resource to CSF4 [7], which is an open source meta-scheduler released as an execution management component of the Globus Toolkit 4 [10]. CSF4 can coordinate heterogeneous local schedulers and provide a uniform and transparent compute resource access interface.

In the past, using CSF4 was fairly complicated – users needed to know details of each of the clusters, such as deployment of applications, location of replicated data, dynamic state of clusters, etc. The resource management of CSF4 paid more attention to the integration of heterogeneous local schedulers [9]. As part of our work, we propose a novel way to virtualize application resources to make CSF4 easy to use via command-line as well as programmatic means. First, we divide grid resources into two parts - application resources and cluster resources. The description of an application resource includes necessary parameters: “name”, “path”, “clusters deployed on” and optional parameters “version”, “compiler” and “dependent libraries”. The application resources themselves are fairly static and do not change frequently once they are deployed – their status can be updated by the local administrator, and queried via MDS [12] or SCMSWeb [26], or even an RSS (Really Simple Syndication) feed.

To submit a job to CSF4, a client just needs to specify the name of the application resource. After the scheduling decision has been made, as described below, the scheduler can use an appropriate version of the application, taking into account the location of binaries and dependencies from the resource information, thus obviating the need for an end-user to specify it each time. This is analogous to our model for the Web services where a user just deals with a service, and is not bothered with the internal details.

The description of cluster resource includes necessary parameters: “name”, “infrastructure type”, “scheduler type”, “master hostname”, “available CPUs” and optional parameters: “scheduler version”, “scheduler port”, “CPU architecture”, “memory”, “disk space”. Some parameters of cluster resources, such as “available CPUs”, are finite and change very frequently -- although some monitoring tools such as Ganglia [11] provide some semi-real-time cluster status information via MDS, this information has been found to be not very dependable since they are not gained from local schedulers and are frequently out of date. Hence, we also maintain an inner resource status list in CSF4, and change the resource status on every scheduling cycle using reports on the status of completed jobs. When a user submits a job to CSF4, it queries the application and cluster resources from a provider.

In the scheduling cycle, CSF4 regards the user job as a set of resource requirements and matches the requirements with resources available via FCFS (first come first server) policies. CSF4 locks the allocated resource by changing the inner resource status list after a successful match. The locked resource will be released after the job finished. CSF4 also adjusts the value of “available CPUs” after the completion of submitted jobs. On one cluster, if most of the dispatched jobs are finished, CSF4 will give a higher weighted value for the “available CPUs” of this cluster. On the other hand, if few jobs are finished, which suggests that the cluster may be busy or that the local scheduler did not allocate all the CPUs for grid jobs – in this case, the value “available CPUs” for the cluster will be weighted down. For transfer of data between nodes, CSF4 uses the GridFTP protocol [13].

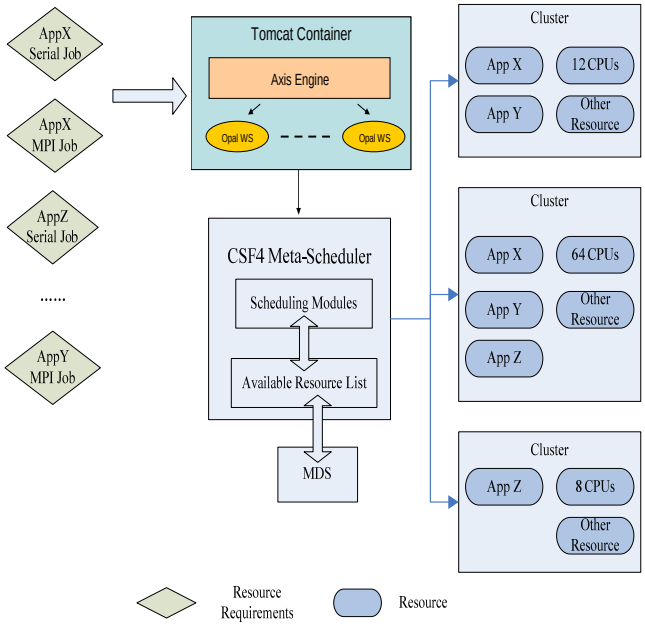

Figure 2 – CSF4 integration with Opal to support scheduling based on resource virtualization

The above figure shows how we have abstracted CSF4 resource virtualization based scheduling within the Opal Web service interface. As part of the application configuration for Opal, developers would specify the name of the application resource, rather than the binary location on a particular system. Instead of submitting jobs to a local scheduler, Opal can perform the job submissions to CSF4 via its WSRF API. Furthermore CSF4 is now able to handle user data stage-in and stage-out and to create

temporary working directory on the execution host using GridFTP transfer protocol [13]. On the other side clients still use the same WSDL API to launch jobs and query status; however, they would be completely oblivious of the complex grid setup at the back-end.

5. The Big Picture

In a complete end-to-end system, we have the grid resources where the applications have been deployed and registered with CSF4. The applications are exposed as Web services using Opal, and accessed via a dynamic Opal GUI as shown in Figure 3.

The raw heterogeneous resources, displayed in the bottom, are accessed via the CSF4 scheduler. As described earlier, CSF4 provides a unified view over the underling infrastructure, exposing only the available applications as resources. The Opal service wraps such an application as a Web service, and hides all the lower level protocols involved with the CSF4 job submission (WSRF, GridFTP, etc). Finally the Opal GUI allows the automatic generation of submission forms for every application, without requiring any extra programming effort. All the data transfers required are handled through Opal and CSF4 coordination, and is completely transparent to the end-users. The integration of Opal and CSF4 furthers the goals outlined in the My WorkSphere paper [24], and is going into production use in the Avian Flu Grid project [25].

Workflows are beyond the scope of this paper however our infrastructure can be orchestrated by any third party workflow systems which support standard Web Service protocol. For example BPEL [28] can directly leverage application level interfaces exposed by Opal. Nevertheless, if supported by the workflow engine or plug-in, orchestration can also be implemented at the CSF4 level, which adopts plain WSRF [8] protocol.

6. Related Work

There are several efforts that address dynamic deployment of Graphical User Interfaces (GUI). For example, the Mozilla framework uses the XML Markup Language (XUL) [2] for defining a rich user interface portable over different platforms. Its syntax includes all the most common graphical widgets like text box, check box, file chooser, etc. The key difference of our approach from this is that we do not define any presentation logic in our application configuration. Our goal is that this is flexible enough to be converted into different presentations – indeed, it is conceivable that our XML configuration can be translated into XUL via XSLT transforms.

PISE, an application to create web based user interfaces [3], allows specification of command line parameters using an XML based configuration file. Thus with a set of PERL scripts, PISE is able to create web forms and CGI scripts for the execution of applications. Its complex configuration file, which also includes PERL statements, enables definition of advanced rules for command line arguments validation. However its complexity can restrict application developers from adopting this solution. The main drawback of PISE is its static interface generation engine, which can create submission forms only at compile time, while the Opal GUI is able to dynamically discover application parameters and build submission forms [31].

Soaplab [5] is a tool that allows users to expose applications via the Web service protocol. It uses a format called ACD to define the parameters required to run the program, which allows a fine grained specification of input data type (integer, float, string, etc.),

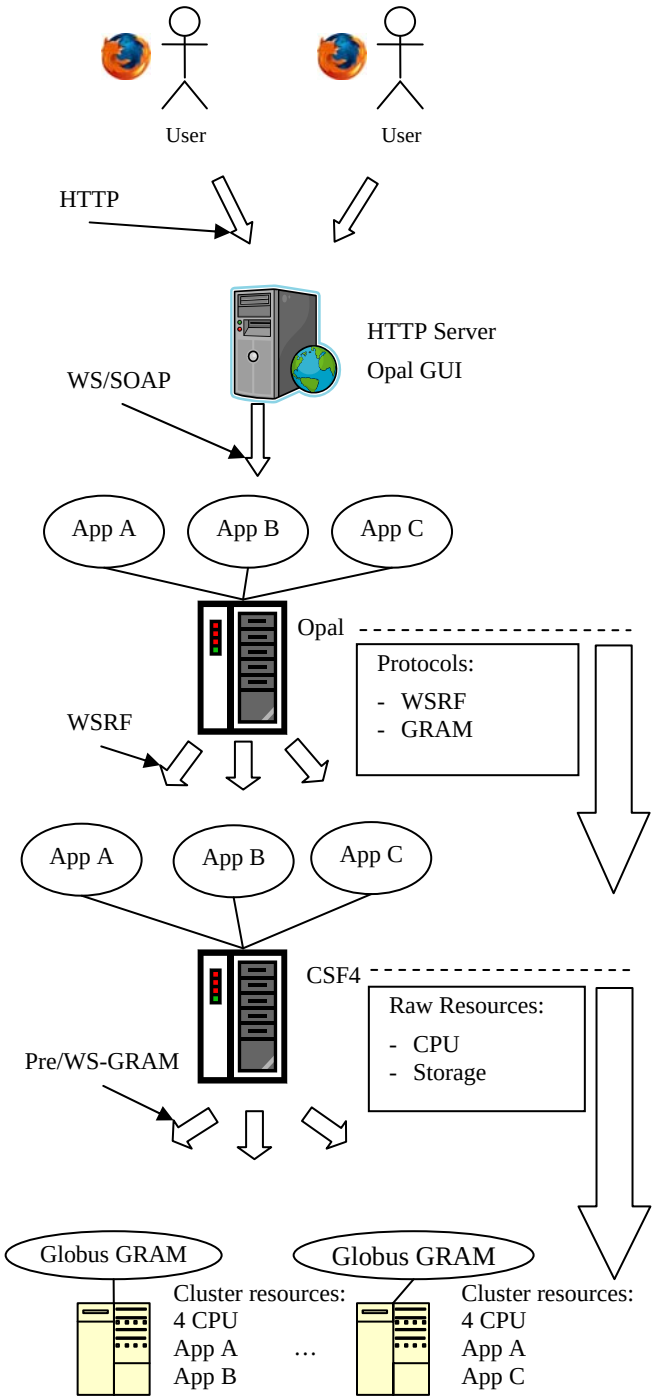

Figure 3 - NBCR infrastructure, CSF4, Opal, and Opal GUI deployment

different input parameters. It uses CORBA [6] in the back-end to access available applications – apart from being another software tool to install. This is also a concern because CORBA is a non-

standard tool in the grid world. Moreover, Soaplab does not provide support for any grid authentication and access control (GSI [16]). However, one interesting feature of ACD language is its capability to allow specification of semantically related constrained parameters and default values. This functionality can be useful in cases where default values can vary depending on user selection made up to that point. At this point, the Opal deployment configuration file does not allow specification of such complex validation rules; on the other hand, it makes it much easier to use.

6.1 Conclusions

In order to simplify usage of grid infrastructures and to increase their adoption, one has to provide easy to use abstractions to complex lower level constructs. For this reason, we have provided a mechanism to model applications as resources within the CSF4 meta-scheduler framework. Furthermore, by wrapping applications as Web services via the Opal toolkit, we have provided an application-centric view of the grid to Web services clients. We have provided an extension to the Opal toolkit to describe command-line arguments, which can be used to dynamically create Web based submission interfaces to these applications, in a Web 2.0-like manner. We believe that providing such easy-to-use mechanisms would lower the entrance barrier for end-users to effectively use grid middleware.

7. ACKNOWLEDGMENTS

The authors would like to acknowledge support from the NIH P41 RR 08605 to NBCR; TATRC W81XWH-07-2-0014 to PWA, the Gordon and Betty Moore Foundation grant to the CAMERA project, and the NSF Grant No. INT-0314015 and OCI-0627026 for the PRAGMA project.

8. REFERENCES

[1] Krishnan, S., Stearn, B., Bhatia, K., Baldridge, , K. K., Li, W. W., and Arzberger, P. 2006. Opal: SimpleWeb Services Wrappers for Scientific Applications. IEEE International Conference on Web Services (ICWS 2006), pp 823-832.

[2] The Mozilla Corporation. XML User Interface Language (XUL). http://www.mozilla.org/projects/xul/ .

[3] Letondal, C. 2001. A Web Interface Generator for Molecular Biology Programs in Unix. Bioinformatics, 17(1), pp 73-82.

[4] Dolinsky, T. J., Nielsen, J. E., McCammon, J. A., and Baker, N. A. 2004. PDB2PQR: an automated pipeline for the setup, execution, and analysis of Poisson-Boltzmann electrostatics calculations. Nucleic Acids Research, 32, W665-W667.

[5] Senger, M., Rice, P., and Oinn, T. 2003. Soaplab—A Unified Sesame Door to Analysis Tools. Proceedings of the UK e-Science All Hands Meeting.

[6] CORBA/IIOP Specification, Object Management Group, Inc., March 4, 2001 see http://www.omg.org/technology /documents/formal/corba_iiop.htm .

[7] Wei, X., Ding, Z., Yuan, S., Hou, C., and Li H. 2006. CSF4: A WSRF Compliant Meta-Scheduler. International Conference 2006 on Grid Computing and Applications, Las Vegas, USA.

[8] Czajkowski, K., Ferguson, D. F., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe W. 2004. The WS-Resource Framework. March 5.

[9] Xiaohui, W., Zhaohui, D. 2006. GDIA: A Scalable Grid Infrastructure for Data Intensive Applications. 2006 IEEE International Conference on Hybrid Information Technology, 9-11 Nov.

[10] Foster, I. 2006. Globus Toolkit Version 4: Software for Service-Oriented Systems. Lecture Notes In Computer Science, vol. 3779, pp. 2-13.

[11] Massie, M., Chun, B. and Culler, D. 2004. The Ganglia Distributed Monitoring System: Design, Implementation and Experience. Parallel Computing, vol. 30, pp. 817-840.

[12] Zhang, X. and Schopf, J. 2004. Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2. Proceedings of the International Workshop on Middleware Performance (MP 2004), part of the 23rd International Performance Computing and Communications Workshop (IPCCC), April.

[13] Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., and Foster, I. 2005. The Globus Striped GridFTP Framework and Server. Proceedings of Super Computing 2005 (SC05), November.

[14] Li, W. W., Baker, N., Baldridge, K., McCammon, J. A., Ellisman, M. H., Gupta, A., Holst, M., McCulloch, A. D., Michailova, A., Papadopoulos, P., Olson, A., Sanner, M. and Arzberger, P. W. 2006. National Biomedical Computation Resource (NBCR): Developing End-to-End Cyberinfrastructure for Multiscale Modeling in Biomedical Research. CTWatch Quarterly, Volume 2, Number 3, August.

[15] Bailey, T. L., Williams, N., Misleh, C., and Li, W.W. 2006. MEME: discovering and analyzing DNA and protein sequence motifs. Nucleic Acids Res 34, pp. 369-373.

[16] Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., and Tuecke, S. 1003 Security for Grid services. High Performance Distributed Computing, proceedings, pp. 48-57, 22-24 June.

[17] Krishnan, S., and Bhatia, K. 2007. SOAs for Scientific Applications: Experiences and Challenges. Accepted to the 3rd International Conference on e-Science and Grid computing, December.

[18] The GridSphere Portal Framework – http://www.gridsphere.org.

[19] O'Reilly, T. 2005. What is Web 2.0. http://www.oreilly.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[20] The Gemstone Rich Internet Application. http://gemstone.mozdev.org

[21] MGL Tools – http://mgltools.scripps.edu

[22] Li, W. W., Krishnan, S., Mueller, K., Ichikawa, K., Date, S., Dallakyan, S., Sanner, M. F., Misleh, C., Ding, Z., Wei, X., Tatebe, O., and Arzberger, P. W. 1006. Building Cyberinfrastructure for Bioinformatics Using Service Oriented Architecture. In the 6th IEEE International Symposium on Cluster Computing & the Grid, Singapore, p. 39.

[23] Bhatia, K., Chandra, S., and Mueller, K. 2005. GAMA: grid account management architecture. E-Science and Grid Computing, pp. 8, December.

[24] Ding, Z., Luo, Y., Wei, X., Misleh, C., Li, W. W., Arzberger, P. W., and Tatebe, O. 2006. My WorkSphere: Integrative Work Environment for Grid-unaware Biomedical Researchers and Applications. In Supercomputing Conference 2006, SC06, 2nd Grid Computing Environment Workshop, Tampa, Florida.

[25] Avian Flu Grid - http://avianflugrid.pragma-grid.net.

[26] Uthayopas, P., Maneesilp, J., and Ingongnam, P. 2000. SCMS: An Integrated Cluster Management Tool for Beowulf Cluster System. in PDPTA, Las Vegas, Nevada, USA

[27] Continuity – Cardiac Mechanics Research Group, UCSD - http://www.continuity.ucsd.edu/

[28] Jordan, D., and Evdemon, J., 2007. Web Services Business Process Execution Language Version 2.0. OASIS Standard, 11 April.

[29] Kropp, A., Leue, and C., Thompson, R., 2003. Web Services for Remote Portlets Specification Version 1.0. OASIS Standard, August.