Accelerating Science Gateway Development with Web 2.0 and Swift

Wenjun Wu
Computation Institute, University of Chicago & Argonne National Laboratory
Chicago, IL 60637, USA
wwj@ci.uchicago.edu

Thomas Uram
Mathematics and Computer Science Division, Argonne National Laboratory
Argonne, IL 60439, USA
turam@mcs.anl.gov

Michael Wilde, Mark Hereld, Michael E. Papka
Mathematics and Computer Science Division, Argonne National Laboratory
Argonne, IL 60439, USA
wilde, herald, papka@mcs.anl.gov

ABSTRACT
A Science Gateway is a computational web portal that includes a community-developed set of tools, applications, and data customized to enable scientists to run scientific simulations, data analysis, and visualization through their web browsers. The major problem of building a science gateway on a Grid environment such as TeraGrid is how to deploy scientific applications rapidly on computational resources and expose these applications as web services to scientists. Although many web-service frameworks have been designed and applied in building domain-specific science gateways, most of these efforts only addressed the issue of adding scientific applications as SOAP services into a service container; they usually don't provide solutions to support web interface generation. Developers still need to spend a lot of time learning web programming to implement a user-friendly and interactive web interface to these services.

In this paper we propose a new application framework that can greatly accelerate the development cycle of science gateway systems. This framework enables science gateway developers to import their domain-specific scientific workflow scripts and generate Web 2.0 gadgets for running these application workflows and visualizing the output from workflow executions without writing any web related code. By assembling these application-specific gadgets and some common gadgets predefined in the framework for workflow management, developers can easily set up a customized computational science gateway to meet community requirements. We demonstrate the utility of the framework with an example from computational biochemistry.

Categories and Subject Descriptors
H.3.5 **[Online Information Services]:** Web-based services

General Terms
Design, Experimentation

Keywords
Web2.0, Workflow, Science Gateway, OpenSocial

1. INTRODUCTION
This paper proposes a new Web 2.0 based application framework that provides a hosting environment for running computational workflows and delivers user-defined workflows as both web services and OpenSocial [6] gadgets. Through this application framework, science gateway developers can focus on defining computational workflows for domain-specific applications using parallel scripting technology [7], and can use the software tools in the framework to automatically generate gadgets for running the workflows and visualizing the outputs from workflow executions. By assembling these application-specific gadgets and some common gadgets predefined in the framework for workflow management, developers can easily set up a customized computational workspace to meet community requirements.

A science gateway is a computational web portal that includes a community-developed set of tools, applications, and data customized to meet the needs of a targeted community. It can hide the complexity of accessing heterogeneous Grid computing resources from scientists and enable them to run scientific simulations, data analysis and visualization through their web browsers. It is also a collaborative cyber-environment on which researchers working on the same or similar domains can easily team up to perform computational thinking on challenging scientific problems by sharing their computational software tools and elevating experimental datasets to scientific knowledge and innovative theories. Initiated in 2004, many scientific gateways funded by the TeraGrid Science Gateways program [1] have been developed to offer Software-as-a-Service (SaaS) to researchers from science domains such as bioinformatics, climate, and high-energy physics.

The gateway paradigm requires gateway developers to compile and install scientific applications on a variety of HPC clusters available from the resource providers in TeraGrid, to build service middleware for the management of the applications, and to develop web interfaces for delivering the applications to a user's web browser.

One major problem in the development of a gateway is how to rapidly deploy scientific applications on computational resources and expose these applications as web services to scientists. Clearly, the whole process of application integration can be greatly streamlined if a generalized application framework can facilitate the development procedure. Although many web-service frameworks [2][3] have been designed and applied in building domain-specific science gateways, most of these efforts only addressed the issue of adding scientific applications as SOAP services into a service container. These frameworks also enable

workflow orchestration based on the deployed web services [4]. But they usually don't provide solutions to support web interface generation. Developers still need to spend a lot of time learning web programming, especially JavaScript and AJAX technologies to implement a user-friendly and interactive web interface to these services

Another issue regarding application integration is how to host user defined workflows in a science gateway. Scientific workflows or computational workflows have become an effective programming paradigm to compose complex computation modules for scientific simulation and analysis of large-scale datasets on clusters or heterogeneous computing environments. For instance, a data-driven workflow language such as Swift [5] defines a high-level abstraction and model for computational software developers to exploit the concurrency and parallelism of their data-intensive science applications. Given large-scale data inputs, a computational workflow usually requires a lot of computing resources and a very long time to run tasks. A workflow hosting environment greatly facilitates developers to easily manage their workflows, reliably run the workflows, profile the performance of their workflows, and expose the workflows as services to their user communities.

The rest of the paper is organized as follows. Section 2 presents an overview of the Web 2.0 application framework. Section 3 and Section 4 describe the design and implementation of the application management service and the workflow execution service of the framework, respectively. Section 5 discusses the design of the Web 2.0 representation layer and shows a science gateway example for protein 3D structure prediction based on our framework. Security issues around the framework are examined in Section 6. Related work is discussed in Section 7. Finally, we give the conclusion in Section 8.

2. WEB 2.0-BASED SCIENTIFIC APPLICATION FRAMEWORK

We selected OpenSocial as the basis for our Web 2.0-based scientific application framework. OpenSocial [6], a social networking framework initiated by Google, presents a Web 2.0 approach to the integration of web applications and the construction of collaborative cyber environments. It standardizes the practices of both gadget programming and online social networking, enabling web developers to write social gadgets that can run in any OpenSocial-compliant container. In OpenSocial, web applications are regarded as gadgets, which define their own HTML content and control logic in client-side JavaScript. Such a self-containing gadget has less dependence on its container than a portlet [8], which simplifies the deployment of the gadget. OpenSocial also provides a social data API to access information about people, their friends, and their data, within the context of an OpenSocial container.

The other key building block in the framework is the workflow system. Swift [7], a parallel scripting language for developing data-intensive workflow applications is the only workflow language supported in our framework. The major advantages of supporting Swift in the framework include: (1) the Swift language blends a C-like syntax with functional programming characteristics, which is designed to expose opportunities for parallel execution; (2) Using the Swift language, developers can easily link their scripts into a data-driven computational workflow; and (3) the Swift workflow engine is very lightweight for integration into a science gateway.

Figure 1 shows the basic structure of this Web 2.0 based application framework. Like most service-oriented architectures, this is a three-layer system: the resource layer including distributed computing resources and science data repository; the service layer consisting of application management service (AMS), workflow execution service (WES) and science data service (SDS); and the top web representation layer based on OpenSocial gadget technology.

The application management service enables developers to describe the command-line syntax, the output dataset, the dependence on installed software packages, and the run-time requirements of their workflows. These metadata and Swift workflow packages are kept in a science application registry. Parsing the metadata of a Swift workflow, the AMS generates gadgets for launching the workflow and visualizing its output. The workflow execution service is built on top of the Swift workflow engine, which can run Swift workflows on Grid environments through Globus GRAM [9] and SSH, and also supports multiple data access methods (e.g. local file system, sftp and GridFTP [10]). The WES allows users to start, stop, and restart Swift workflows that have been published in the science application registry, and to monitor the progress of the workflow execution. The science data service provides a unified RESTful interface to support basic file operations and query and annotation of heterogeneous data resources. Through the SDS, the WES entity can make data queries to discover the real locations of input and output data for running a workflow and pass the locations to a workflow executor. A variety of data gadgets can also be developed on the basis of the SDS for end users to find, access, and annotate their data.

The following two sections present more discussion about the design of the AMS and WES. Although there are some common software patterns in providing science data services, the design of the SDS is often deeply affected by the application requirement and data ontology of a specific science domain. Therefore this framework only assumes that the SDS should provide a unified

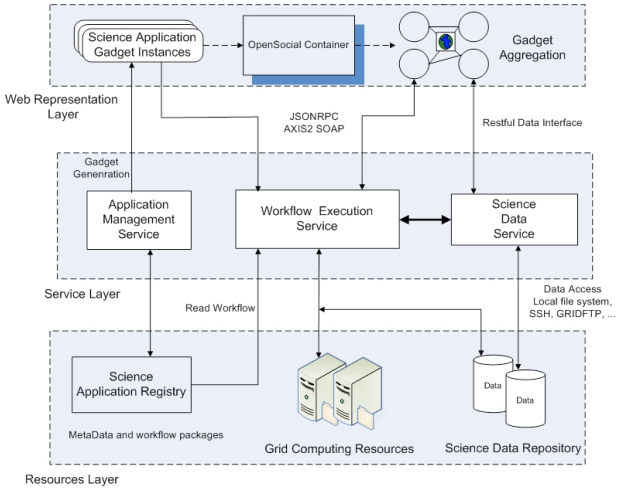


Figure 1. Architecture of the Web2.0 application framework

RESTful interface to other components, and leaves the detailed design of the data model up to the implementation of specific gateway projects.

3. SCIENTIFIC APPLICATION MANAGEMENT

In this section we present an example of using this Web 2.0 workflow framework to define applications, create application gadgets, and run workflows. This example comes from an important computational problem in biochemistry: protein 3D structure prediction. University of Chicago researchers have developed the Open Protein Simulator (OOPS) [11] for predicting tertiary (3D) protein structure. Their approach to this computational problem involves running many instances of a structure prediction simulation, each with different random initial conditions. The simulation uses an "iterative fixing" algorithm that performs multiple "rounds," each involving many parallel Monte Carlo simulated annealing models of molecular moves with energy minimization. Figure 2 shows the main function defined in the OOPS workflow script.

```
main()
{
string plistfile=@arg("plist",""); // input protein
fasta file
string indir=@arg("indir","oops.input");
string outdir=@arg("outdir","output");
string nsims=@arg("nsims","1"); // simulation num
string st=@arg("st","100");    // start temperature
string tui=@arg("tui","100"); // time update interval
string coeff=@arg("coeff","0.1");

string plist[] = readData(plistfile);
RAMAIn ramain[]
 <ext;exec="RAMAInProts.map.sh",i=indir,p=plistfile>;
RAMAOut ramaout[][]
        <ext;exec="RandProtRadialMapper.py",
                o=outdir,p=plistfile,
                n=nsims,c=create>;
foreach sim in [ 0 : @toint(nsims) -1 ] {
  foreach prot,index in plist {
    ramaout[index][sim]
      =predictCf(prot,ramain[index],st,tui,coeff);
    VizOut outpng[] <ext; exec="pngmapper.py",
      o=metadir,
      p=@filename(ramaout[index][sim].pdb) >;
    outpng[0] = pngviz(ramaout[index][sim]);

  }
}
}
```

Figure 2 . OOPS protein simulation workflow script

In this protein simulation workflow script, the command line would be executed as follows:

```
swift oops.swift -plist=plist -nsims=1 \
     -st=100 -tui=10 -coeff=0.1"
```

Such a command-line syntax can be clearly described in Mobyle XML [12], whose schema can specify the type, name, and format of each argument in a flexible way. Moreover, short Python or Perl code snippets can be inserted into a Mobyle description to capture the dependence among the arguments and generate the appropriate argument format. Figure 3 displays a fragment of the Mobyle XML description for OOPS simulation.

```
<parameter ismandatory="1" issimple="1" ismaininput="1">
    <name>plist</name>
    <prompt lang="en">input protein fasta file</prompt>
    <type>
            <datatype>
            <class>File</class>
            </datatype>
     </type>
     <format>
             <code proglang="python">
            ("","-plist="+str(value))[value is not None]
            </code>
       </format>
    <argpos>1</argpos>
  </parameter>
```

Figure 3. Mobyle XML description for OOPS simulation workflow script

Parsing the Mobyle XML description, the application gadget generator in the AMS can create an OpenSocial gadget that consists of the gadget metadata, as well as HTML markups and JavaScript codes by using predefined XSLT templates and JavaScript templates. These templates transform the Mobyle XML into an HTML snippet with predefined cascading style sheets to produce JavaScript codes for security handling, parameter marshalling, and invocation of the generic JSON-RPC [13] service, which calls the WES to launch OOPS simulation runs. Figure 4 shows a screenshot of the OOPS simulation gadget as

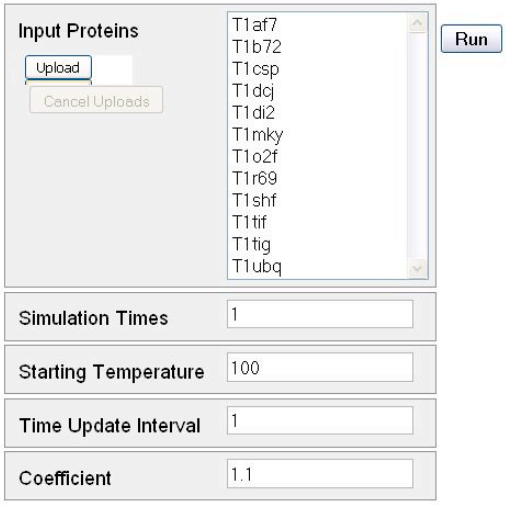


Figure 4 OOPS simulation gadget as generated from Mobyle XML by the application management service

generated from the Mobyle XML.

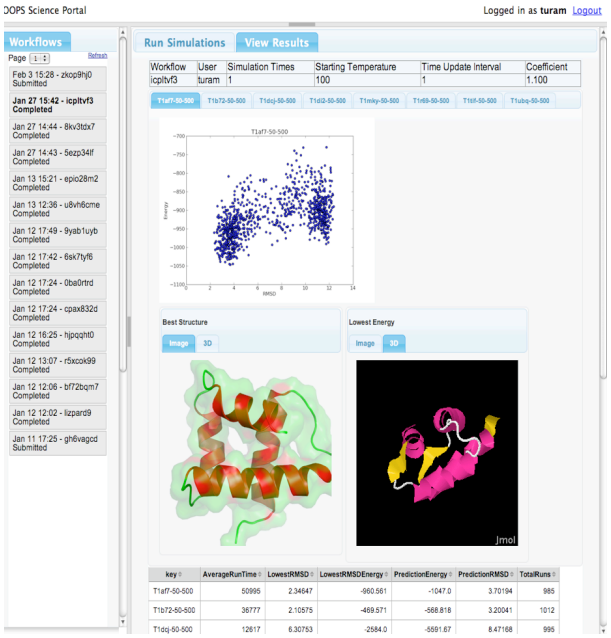
The output from this protein simulation workflow includes predicted protein 3D structures in the standardized PDB format and the data analysis result for the evaluation of prediction quality. These output files are visualized through a range of visualization tools such as PyMol [14] for generating PNG images of protein structures, and scatter plots of protein energy level versus the root-mean-square distance of backbone atoms of the predicted structure to the known structure. These analysis computations are accomplished as a final step in the integrated simulation script. Clearly, visualization of the outputs from workflows is highly dependent on the requirement of domain-specific applications. We sought to allow workflow developers to explicitly describe their visualization methods and render the visualization results in a gadget while minimizing the HTML and Javascript code that must be written. To this end, we took a data-driven approach in the framework, in which workflow developers include analysis processing in their workflows and, finally, specify the output data to be visualized. Figure 5 shows an XML fragment describing the output of the OOPS workflow.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<output>
   <file label="Data Summary">summary.csv</file>
   <group name="T1af7-25-100">
     <file label="">
       T1af7-25-100_scatter.png
     </file>
     <file label="Best Structure">
       T1af7-25-100_best.pdb
     </file>
     <file label="Lowest Energy">
        T1af7-25-100_predicted.pdb
     </file>
  </group>
</output>
```

Figure 5 XML description for the output of OOPS simulation workflow

The above XML description is very straightforward. It lists all the output files to be displayed as visualization results and groups files into a bundle if they are produced from the simulation runs on the same input protein. In addition, for each science application, developers can define an XSLT rule file to transform the XML file into an HTML. These XSLT templates specify the HTML objects for rendering various types of output files and layout schemes based on the grouping of the data files. The visualization page for the Figure 5 output XML is shown in Figure 6.

In addition to software tools for describing the command-line interface and visualization processing of a workflow, the AMS also allows workflow developers to keep multiple versions of their workflow scripts. During the development and testing of workflows, developers may find it useful to create different versions of Swift scripts for the same computing task. Each script could have different input and output arguments. Thus we must generate gadgets for these scripts if workflow developers want to evaluate all the versions to find out the best solution to their computational problems.



Figures 6 The Main Page of OOPS Science Gateway showing the workflow history and visualization result

4. WORKFLOW EXECUTION SERVICE

The WES is implemented in two major components: one is the service-facing module that provides JSON-RPC and AXIS-2 web services, the other is the Swift engine daemon that actually launches Swift workflow instances. Whenever a user sends a workflow request through the service interface, the request is first processed to generate a command-line for workflow launching, and then recorded in a persistent database that is polled by the Swift engine daemon. The daemon executes workflows by forking Swift workflow executor processes that run Swift workflow stacks with the command line carried by the requests. Because this workflow executor has a built-in weighted scheduling mechanism for the task execution, it can dynamically decide the best Grid resources that are available to run the Swift workflow and can resubmit any failed tasks. Based on the log files produced by the executor during the execution of the workflows, the Swift daemon monitors the dynamic progress of the workflow and updates its status in the workflow database.

The WES relies on running environment configurations that are kept in the application registry to kick off the execution of a workflow application. Such a configuration typically defines a list of all the executable programs, their installation locations, security contexts and relevant environment variables and attributes. Workflow developers must provide this configuration when they publish a new Swift workflow application through the AMS service. The other dependence of the WES is on the data access mapping service from the SDS. Workflow invocation requests only carry URL references to the input datasets for workflow running. Thus, the WES has to ask the SDS for the physical

addresses of the data and appropriate data access methods, and pass the information down to the Swift engine daemon.

Elastic service provision is one of the major design issues in the workflow execution service. For the resource provision of running computing workflows, the WES can integrate resource reservation provided by third-party scheduling services developed by Grid communities and the built-in resource provider (Coaster) in the Swift workflow engine [15]. Swift Coaster implements a similar provision mechanism to Condor Glide-In [16] to improve the throughput of job submission and execution for many-tasks workflows. Furthermore, besides the resource provision for running remote workflow tasks, the WES also needs to handle load balancing for running many workflow executor processes on local hosting environments for science gateways. Driven by the scale of the input dataset, a single workflow executor can be a long running process that can consume a few Gigabytes memory and 5%-10% CPU on a modern web server. Since the functional components of the WES has been divided by the persistent workflow storage, it is reasonable to implement a simple clustering mechanism to run a lot of workflow executors on a modest server farm to cope with the dynamic workload for a science gateway. For systems with a very large user community and higher workload, it is also feasible to run workflow executor processes as computing tasks in a computational cluster.

Workflow and data provenance is another very important feature for scientists and researchers when they are performing large-scale simulation and analysis in an iterative manner. Sometimes they want to run new workflows based on the output of finished workflows and track the lineage of these workflows later to explore the best parameter set among them. The WES tracks provenance by annotating the workflow database record with the lineage information when users select the output from completed workflow runs as the input data to launch a new workflow. The WES allows users to search a group of workflow runs that are generated along a lineage history.

5. WEB 2.0 DELIVERY LAYER

All the service entities including AMS, WES and SDS deliver their services as OpenSocial gadgets to users. Since an OpenSocial gadget is a standalone client-side web application that can be rendered on any OpenSocial compatible container, we should decide whether to host an in-house container at the web representation layer of our framework. The major benefit of running an in-house container over a commercial one hosted by social networking web sites is the ability to customize security policy and gadget layout.

Generally, scientists and researchers are open to share their data and workflow tools with the public. However there are scenarios where data security becomes a very important concern if their studied subjects have privacy protection issues. It is essential, therefore, to set up a private OpenSocial container for rendering scientific gadgets to alleviate such a privacy concern. It is also easier to customize the layout of gadgets and choose the most suitable version of the OpenSocial specification to be used in our framework if we run our own OpenSocial container.

Currently, Shindig [17], an OpenSocial reference implementation supported by the Apache community, is the foundation for the development of this layer. It is worthwhile to note that Shindig is not a full-fledged OpenSocial container because it has no services such as gadget layout, gadget management, and security. We have to build up these services on top of Shindig at the web layer of our framework. Basically the web layer is just like an enhanced gadget container with customized layout and management policy.

Customized layouts make the user interface look more like an integrated web environment, as opposed to the column layout style used in iGoogle. Figure 6 displays such a layout for the OOPS Science Gateway project, which presents a workflow-centered management GUI to OOPS users. In the layout, three gadgets, including the workflow history gadget, the results view gadget, and the OOPS simulation gadget (hidden in tab), are integrated together to become a powerful workbench. Behind the scenes, communication channels are set up among the gadgets and container page to support the event-and-listener model in GUI design.

The OpenSocial specification offers APIs for both publish-subscribe messaging for gadget-to-gadget communication and RPC communication between gadgets and their container page. For example, in Figure 6, the OOPS simulation gadget publishes a message after an invocation of the OOPS workflow is submitted. The workflow history gadget that has subscribed to this message topic will get notified and update its gadget content by placing a new workflow item on the top of the list. At this time, if a user clicks this item in the workflow history gadget, it sends a message to notify the container page to switch to the View Results tab to show the simulation output.

The issue of cross-domain communication in a web browser may arise if gadgets and their container page are rendered from different domains. In some implementations, for example, the container web page runs on port 80, while the gadgets are rendered on port 8080 because Shindig is running on port 8080 as a Tomcat web application. Being loaded in an IFRAME of the container page, a gadget has to follow the JavaScript sandboxing and same-domain security policy, thereby prohibiting its JavaScript code from accessing the container page unless the IFRAME comes from the same domain. The OpenSocial gadget-to-container RPC mechanism addresses this problem by trying possible cross-domain communication APIs within different browsers and presents a clean RPC interface to gadget developers.

Since each gadget is loaded within an IFRAME in the container page, it has no knowledge of web sessions. The only way to allow the gadget to get a session object is to pass a unique session ID to the gadget when the gadget IFRAME is created and then the gadget can call back to the container to query the attributes in the session object using the ID. Each gadget IFRAME has a source URL with an important parameter called security token. This is a short-lived token that has encoded in it all the necessary information about the site, gadget, and viewer. Once a gadget is initialized, it parses the security token in the URL and reads the session ID to contact the container page to fetch the current session object.

6. SECURITY

Within a service framework, security is certainly a critical design issue. There are a variety of security solutions to web-services and Web 2.0 systems, including HTTPS/SSL based transport level security, WS-Security and WS-Trust. These solutions can be separately applied in securing JSON-RPC services and SOAP services when we implement this application framework.

The issue of workflow execution sandbox arises if we allow end users without any mutual trust to publish arbitrary workflows and run OpenSocial gadgets to launch their workflows. To some extent, the sandbox is needed both on the local resource for running Swift workflow executors and on the remote Grid resources where workflow computing tasks are submitted. Virtual machines seem the ultimate solution to providing sandboxes for isolating local workflow executors from different users. The alternative approach is to put some file access restrictions for running these executors. For instance, a workflow instance from a user can only modify and remove data objects owned by this user.

It is also necessary to design some sort of security policy for running remote computing tasks for workflows on community-shared resources. Many science gateways follow the transitive mode mentioned in the "AAA model to support science gateways with community accounts" [18] to enforce access to remote TeraGrid resources. In this model, the key notion is the so-called TeraGrid community account, which is a shared TeraGrid account for serving community users who don't have a formal account in the TeraGrid system. Through its community account, a science gateway service can retrieve the temporary MyProxy [19] credential from the MyProxy server of the TeraGrid, and use that credential for job submission and data transfer. Providing a secured sandbox for such a community account across Grid environments is still an active research area and needs further investigation.

From the perspective of the application deployment in this Web2.0 service framework, we provide a collaborative mechanism between administrators and users to work together in the procedure of application deployment to make sure that only safe applications get deployed in a science gateway. When a user completes a new workflow XML description through the AMS, he can send a request to an administrator requesting installation of the software packages needed by this workflow into the TeraGrid resources. Following the software list of the new workflow, the administrator can download the software packages and install and test them on multiple TeraGrid clusters. After the workflow is successfully deployed, the WES will be activated to make this workflow available for end users.

7. RELATED WORK

Most application frameworks for building science gateways [2][3] are often focused on wrapping scientific applications as web services and delivering the services through SOAP. They also allow workflow developers to utilize workflow composing toolkits to define computational workflows based on the deployed scientific services [4]. Some computational application frameworks rely on Condor middleware [16] for job management and resource scheduling.

The service layer of our framework is more focused on lightweight JSON-RPC services that can provide fast communication channels between front-end AJAX JavaScript codes and backend workflow services. We don't need Condor for resource management because the Swift workflow engine as a standalone package can provide reliable and highly efficient resource management on TeraGrid resources.

TeraGrid Science gateways are using a variety of web programming frameworks for their development, among which the JSR 168/268 based portal framework [8][20] enables users to create a customizable web environment from a collection of application portlets. But a portlet is far less flexible and platform-agonistic than a gadget because it includes both browser-side and server-side components. By using the gadget-based solution, our framework makes it possible for scientists to create their computational web portal on their own social web pages.

8. CONCLUSION

In this paper, we described a new Web 2.0-based application framework that simplifies the development of science gateways. The framework allows developers to host their domain-specific software toolkits and workflows and rapidly generate Web 2.0 service interfaces to their workflows. Thus, it provides researchers a collaborative web environment to run data-intensive computing applications efficiently on Grid resources.

Based on this framework, we have developed the new OOPS science gateway for protein 3D structure prediction. A few OOPS workflows and gadgets have been built and integrated into the science gateway through this framework and have been made available to users. The new science gateway enables researchers to easily utilize petascale systems for exploring a wide range of parameter values and comparing the outcome in order to gain deeper insights into important aspects of the structure and behavioral properties of large biomolecules.

From our experience with building a few science gateways based on this framework, we believe that this web2.0 framework captures a common pattern in the software architecture of science gateways and can be applied to a variety of science domains such as life science, social and behavior science, and scientific visualization. We plan to extend the workflow execution service to support running workflows on the emerging science cloud resources. We will also investigate possible software toolkits to enable the automation of the software development of the whole science gateway based on our current framework.

9. ACKNOWLEDGMENTS

This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357, and the National Science Foundation by grant OCI-0504086.

10. REFERENCES

[1] Wilkins-Diehr, N.,Gannon, D., Klimeck, G., Oster, S., Pamidighantam, S., 2008. TeraGrid Science Gateways and Their Impact on Science, *IEEE Computer* 41(11):32-41, Nov 2008.

[2] Kandaswamy, G., Fang, L., Huang, Y., Shirasuna, S., Marru, S., and Gannon,D. 2006. Building Web Services for Scientific Grid Applications. *IBM Journal of Research and Development* 50(2-3), 2006.

[3] Krishnan, L. and Stearn, B., et al. 2006. Opal: Simple Web Services Wrappers for Scientific Applications. *IEEE International Conference on Web Services (ICWS 2006),* Sep 18-22, Chicago.

[4] Oinn,T., Addis,M. et al. 2004. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics Journal*, 20(17), 3045-3054.

[5] Zhao, Y., Hategan M., Clifford, B., Foster, I., vonLaszewski, G., Raicu, I., Stef-Praun, T. and Wilde, M. 2007. Swift: Fast, Reliable, Loosely Coupled Parallel Computation. In Proc. *IEEE International Workshop on Scientific Workflows 2007*, pp. 199-206.

[6] OpenSocial Specification, http://www.opensocial.org/

[7] Wilde, M., Foster, I., Iskra, K., Beckman, P., Zhang, Z., Espinosa, A., Hategan, M., Clifford, B., Raicu, I. 2009. Parallel Scripting for Applications at the Petascale and Beyond, *IEEE Computer* 42(11):50-61, Nov. 2009.

[8] JSR-168 portlet specification http://jcp.org/aboutJava/communityprocess/final/jsr168/.

[9] Feller, M., Foster, I., and Martin, S. 2007. "GT4 GRAM: A Functionality and Performance Study", *TeraGrid Conference* 2007.

[10] Allcock, W.(editor), GridFTP: Protocol Extensions to FTP for the Grid. GFD-20, April 2003. http://www.ggf.org/documents/GFD.20.pdf.

[11] Hocky, G., Wilde, M., DeBartolo, J., Hategan, M., Foster, I., Sosnick, T. R., Freed, K. 2009. Towards Petascale ab initio Protein Folding through Parallel Scripting, preprint, ANL/MCS-P1645-0609, Argonne National Laboratory.

[12] Mobyle, http://bioweb2.pasteur.fr/projects/mobyle.

[13] JSON-RPC, http://jabsorb.org.

[14] PyMol , www.pymol.org/.

[15] Swift Coaster, http://www.ci.uchicago.edu/swift/guides/userguide.php.

[16] Thain, D., Tannenbaum, T., and Livny, M. 2003. "Condor and the Grid", in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: *Making The Global Infrastructure a Reality*, John Wiley, 2003. ISBN: 0-470-85319-0.

[17] Shindig http://shindig.apache.org.

[18] Welch, V., Barlow, J., Basney, J. and Marcusiu, D. 2006. AAA model to support science gateways with community accounts. *Concurrency and Computation: Practice and Experience* 19(6) 893-904.

[19] Basney, J., Humphrey, M. and Welch, V. 2005. The MyProxy Online Credential Repository. Software: *Practice and Experience*, Volume 35, Issue 9, July 2005, pages 801-816.

[20] Nicklous, M., and Hepper, S. 2008. JSR 286: Portlet specification 2.0. http://www.jcp.org/en/jsr/detail?id=286.