Extending Grid Protocols onto the Desktop using the Mozilla Framework

Karan Bhatia[1] Brent Stearn[1] Michela Taufer[2] Richard Zamudio[2] Daniel Catarino[2]
[1]San Diego Supercomputer Center
University of California, San Diego
9500 Gilman Drive La Jolla, CA 92093
[2]Department of Computer Science
University of Texas, El Paso
500 W. University Ave. El Paso, TX 79902
{karan,flujul}@sdsc.edu {mtaufer,rzamudio,Dcatarino1}@utep.edu

Abstract

This paper describes an approach to building problem solving environments *that provide users with direct access to remote grid services and protocols from their desktop or laptop. We describe two examples of this approach: Topaz extends the Firefox web browser to support the GridFTP protocol, thereby enabling the user to move large files between server-based data repositories and their primary day-to-day computer; and the Gemstone frontend that allows users to* browse *the set of services provided by an application provider and enables dynamic integration of the user interface elements. Gemstone is also a framework that allows application service providers to not only specify the service interface, but define the user interface and experience. Both Topaz and Gemstone build upon the Mozilla platform to leverage the current state of the art in web technologies.*

1 Introduction

Web portals have become a popular end-user environment for many grid computing projects [27, 7, 34, 1]. They provide users with high-level access to services utilizing a heterogeneous set of resources and protocols while shielding the user from the low-level details of their implementations. In addition they require only a web browser to be installed on the user's desktop thereby supporting ubiquitous access. They do, however, require significant server resources including development resources for building the capabilities or customizing the components and maintenance resources for managing the server environment. Ultimately they are limited in the following ways:

- interactivity with the end-user – web portals can only provide as much interaction as allowed by HTML, enough for basic tables and charts, but not enough for scientific visualization.
- leveraging of local resources – web portals can not make use of the significant resources available at the desktop including accelerated graphics cards and hundreds of gigabytes of disk.
- dynamic discovery of new services – web portals require significant development efforts to incorporate new services or capabilities and can not yet dynamically discover or automatically integrate newly created services, a key feature of the emerging grid services standards [18].
- capability to create and integrate new workflows – web portals provide a fixed workflow or set of workflows for managing the execution of well understood sequence of tasks; however, they are not flexible to support exploration of new scientific workflows.

The technology of web portals continues to advance, with additional interactivity provided through the liberal use of JavaScript, Macromedia's Flash, and Microsoft's ActiveX components; however, the benefit of a web portal – the ability to aggregate the services into a single location – is also its fundamental weakness.

Web-based portals are an example of *problem solving environments (PSE)* [23] that are purely server-based. PSEs exist that are purely desktop based as well: the Python Molecular Viewer [4], dataflow-based workflow tools [10, 5, 19], and visualization toolkits [26]. Each has its place among the scientist's toolset, however, each may require recompiles or significant reconfiguration as new services are made available or existing services are updated.

To address these limitations, in this paper we propose an alternative approach that leverages the rapidly advancing technology in the web browser to *extend grid protocols onto the user's desktop*. We describe two examples of this approach: the first extends the popular open-source Firefox web browser to support the GridFTP protocol [9], and the second further expands the approach by building an entirely new Mozilla-based application that provides discovery and access to remote application web services. Section 2 describes Topaz, the GridFTP protocol extension for Firefox, and Section 3 describes the Gemstone frontend that provides access to a set of biomedical applications. Section 4 and 5 discusses related work and concludes.

2 Topaz

The GridFTP protocol is an extension to the standard File Transfer Protocol (FTP) that supports GSI-based security, high performance data transfer using striping and parallel streams, and third party file transfer across different GridFTP servers. The GridFTP server and client tools form an integral part of the Globus Toolkit and provides a basis for higher-level data management capabilities such as the Data Replication Service (DRS) [13].

GridFTP is typically used for data transfer between server systems characterized by high bandwidth and strong host-based security. However, extending the GridFTP protocol to the user's desktop allows the user to easily move data between their primary machine and the traditional server-based grid environments. Any such solution, however, should not require the installation of the Globus Toolkit on the desktop due to its complex installation and configuration. Furthermore, desktops (and laptops) in many environments do not have fixed IP addresses and hence do not meet the strong host security requirements of the Globus Toolkit.

Keeping these requirements in mind, we have developed Topaz as an extension to the popular Firefox browser to integrate the GridFTP protocol on the user's desktop or laptop without the need for other additional grid software. Topaz provides the same semantics to the end-user as is provided by default for the standard FTP protocol. The user provides a gsiftp URL in the form of `gsiftp://hostname:port/path/to/file` by either entering it into the URL bar or by clicking on a link. The Topaz extension prompts the user for security information and contacts the file server using the GridFTP protocol for directory browsing as well as file download and upload. The third-party-transfer functionality is work under development.

The Firefox browser is built on top of the Mozilla framework, the same framework also serves as the basis of the Thunderbird email client as well as other applications. The core framework supports the installation of extensions that can add functionality to the applications – a key feature that has led to the development of a thriving developer community with over 1500 extensions available [3]. Extensions may contain simple user interface additions or extensions to the core framework using the XPCOM (Cross Platform Component Object Module) system [16]. Mozilla-based application extensions follow a uniform standard for packaging and distribution to simplify user installation and updating. Installation of any extension is as simple as selecting a link to the packaged extension or loading it into the browser. Extensions like Topaz require few changes to be used with other Mozilla-based applications.

2.1 Directory Browsing, File Download and Upload

Figure 1 shows the components involved in the Topaz extension. The *GsiFTPHandler* protocol handler is responsible for processing and setting up the data transfer; the *GsiFTPChannel* channel handles data flow between the browser and the stream; and the *GsiFTPStream* stream handles the data transfer between the client and the GridFTP servers through the use of the Globus client libraries that are included with the extension.

For listing directory contents or downloading files, the user passes the gsiftp URL of the GridFTP server to the protocol handler using the Firefox URL bar or selects a gsiftp link. The protocol handler creates an *nsIURL* XPCOM object, a built-in Mozilla component, from the URL string that contains the URL information and the appropriate methods to parse it. The protocol handler also creates a channel for the data transfer and passes it the XPCOM object. The first time the channel is created it generates a login manager to verify that the user has a valid proxy credential. If a valid proxy does not exist, the login manager presents the user with a dialog box to enter a username and password and to select a certificate authority for authentication. Topaz currently only supports the GAMA authentication server [11], but in the future additional credential services. Once authenticated, the GAMA server returns a x509 proxy certificate that Topaz stores in the filesystem within the Firefox profile. Once the security context is created the channel creates a stream and returns a pointer to the stream object. The Mozilla framework uses a sophisticated multi-threaded I/O model and when ready *pumps* the stream until end-of-file is reached.

File upload in Topaz involves less interaction with the Mozilla infrastructure but adds more user interaction. Topaz adds an "Upload File" menu item to Firefox's main "File" menu. When selected, this item calls a JavaScript function to begin the upload sequence: (1) the current browser location URL is retrieved to use as a destination, (2) a *Gsiftp-*

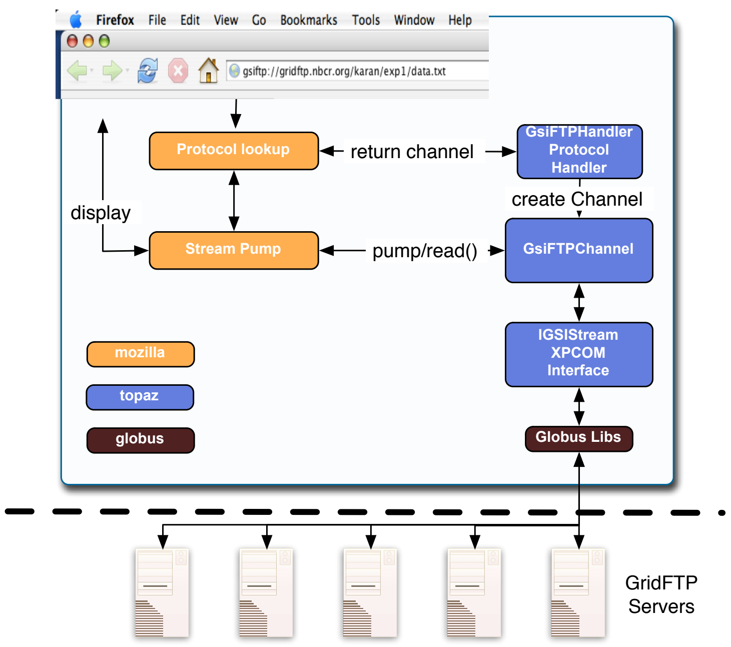

Firefox File Edit View Go Bookmarks Tools Window Help
gsiftp://gridftp.nbcr.org/karan/exp1/data.txt
Protocol lookup
return channel
GsiFTPHandler Protocol Handler
create Channel
display
Stream Pump
pump/read()
GsiFTPChannel
IGSIStream XPCOM Interface
mozilla
topaz
globus
Globus Libs
GridFTP Servers

Figure 1. The Components of the Topaz Firefox Extension.

Stream component is created and (3) passed the URL string; the stream launches a file-chooser dialog to select the file to upload and finally (4) starts the transfer.

2.2 Performance

In this section, we present some initial performance results comparing the performance of Topaz with other GridFTP tools and with other common data transfer tools. The comparisons are not meant to be exhaustive and no protocol optimizations have been performed.

These preliminary results indicate that the additional overhead of Topaz over command line Globus tools is fixed as a function of the file size – Figure 2.a shows a fixed overhead of roughly 1.9 seconds for download and .7 seconds for upload. We believe that this overhead may be reduced by leveraging Mozilla's asynchronous I/O with the parallel data transfer capability of GridFTP; however, for most users this overhead seems acceptable.

Figure 2.b also shows a comparison of data transfer times for non-grid protocols. Specifically, we compared the transfer times in seconds for downloading and for uploading data between a client at the University of Texas at El Paso and a GridFTP server at the San Diego Supercomputer Center using Topaz, the globus-url-copy command in Globus 4.0.1, the Secure Copy Protocol command scp, and the https protocol. We used a set of files with different sizes (i.e., 32KB, 256KB, 2MB, 16MB, 128MB, and 1024MB) and we ran the different download and upload experiments three times for each protocol and each file size. The results show that while the grid protocols are slower than the non-grid protocols, as the file size grows, the differences become negligible. The network bandwidth becomes saturated quickly and the transfer times scale linearly.

We reiterate that no GridFTP protocol optimizations are currently used in these initial experiments, only default settings. For more details and commentary on performance tests see [36].

While we are working to improve the performance, it should be noted that it may be preferable to use the grid-based data movement for reasons other than performance. For example, using scp may involve managing accounts whereas using GridFTP tools will leverage the user credential. In addition, using a non-grid protocol to handle data movement from the desktop may necessitate staging the file on an intermediary server node. Hence, two file copies will be needed as opposed to a single one with GridFTP tools. Finally, GridFTP also supports third-party-transfer, that is, the capability of moving files from one GridFTP server to another in a single data movement as opposed to first staging the data on the local coordinating machine. While this capability is currently in development in Topaz, it can greatly increase performance by not using any significant bandwidth of the client as compared with non-grid protocols.

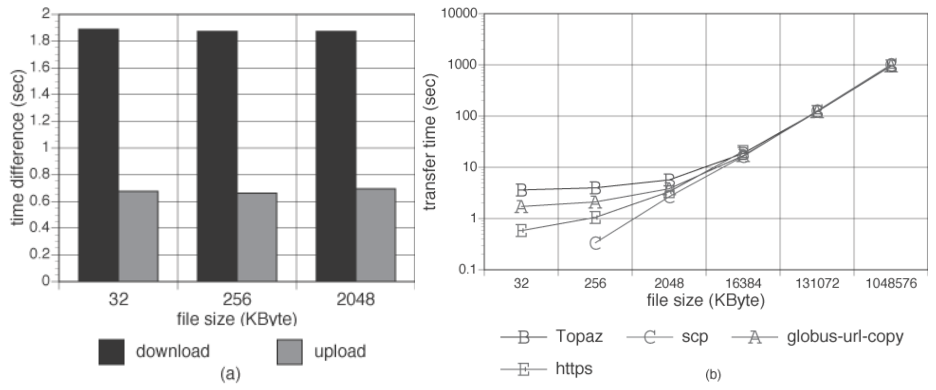


Figure 2. Initial performance data shows (on the left) the additional overhead of Topaz over the Globus command line tools as a function of the data files and (on the right) a comparison of the transfer times for Topaz, Globus command line tools, scp, and HTTPS.

3 Gemstone

While Topaz extends the functionality of the Firefox browser, the Gemstone frontend goes even farther by building an entirely new application on top of the Mozilla source code in the same manner as used to build the Firefox browser and the Thunderbird mail client. The Gemstone frontend is an complete PSE that runs on the user's desktop and provides an interface to accessing remote application web services. Gemstone is also a framework in that it allows application providers to define the user interaction with their applications and provides an architecture and a set of APIs to enable them to do so. The Gemstone framework dynamically integrates the application user interfaces into the frontend when needed, thereby allowing the application provider to evolve the interface as needed without any reconfiguration or plug-ins required by the users.

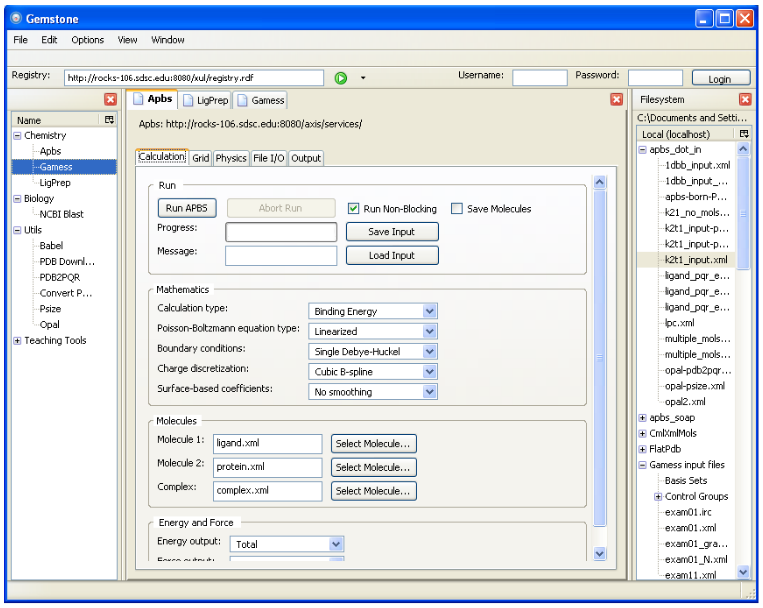
Figure 3 shows the Gemstone frontend interface. The top panel shows a URL bar that points to a particular registry. User's can *browse* different registries of services. For example, the default registry shows the services available by the NBCR services infrastructure [25] hosted at the San Diego Supercomputer Center, but we envision that many centers will publish their service registries in the future. Section 3.1 discusses the registry in more detail.

The far left panel in Figure 3 is the *service list* that lists the services that are published in the registry. Each service entry represents a web service specified by a valid web service endpoint and described by a WSDL document. The services can be grouped into domains and can specify the level of security required for invocation: for example, the Siesta application provided by the NBCR services requires acceptance of a license agreement and authenticated access. We discuss the security capabilities in Section 3.2.

To access a service, users select the service in the service list and the Gemstone framework loads the corresponding *service panel* for that service into the center panel in Figure 3. The service panel provides the user interface for the application, including support for attaching input or data files, setting various application parameters, and retrieving and visualizing the outputs. The service panel is specified in the XML User-interface Language (XUL) [15] which is standard for all Mozilla-based applications. The business logic for the user interface is specified in JavaScript. The Gemstone frontend provides a standard API for accessing the remote services, accessing local resources, managing remote jobs and communicating between service panels. Section 3.3 describes the capabilities of the service panels in more detail.

Finally, the right panel in Figure 3 shows a view into the local directory and supports drag-and-drop across into the service panels. The files in the panel can be edited or visualized depending on their content type. While currently only local file directories are supported, remote GridFTP-based filesystems will also be possible once the Topaz extension is integrated into the Gemstone frontend.

Operating behind the scenes of the Gemstone frontend is a common data model specified in XML Schema that defines datatypes such as molecules, atomic co-ordinates, basis sets, etc. These same definitions are used in the suite of application web services developed in collaboration with NBCR and we are actively working with the application developers within the scientific community to standardize on these definitions. With these common definitions across the set of application web services, the Gemstone frontend supports *exploratory workflows* where users start with a data object, locate all the various operations that could be applied to it, apply a desired operation resulting in a new or



Registry: http://rocks-106.sdsc.edu:8080/xul/registry.rdf

Apbs: http://rocks-106.sdsc.edu:8080/axis/services/

[✓] Run Non-Blocking

Calculation type: Binding Energy
Poisson-Boltzmann equation type: Linearized
Boundary conditions: Single Debye-Huckel
Charge discretization: Cubic B-spline
Surface-based coefficients: No smoothing

Molecule 1: ligand.xml
Molecule 2: protein.xml
Complex: complex.xml

Energy output: Total

Figure 3. The Gemstone framework provides security and discovery of services, but allows application providers to specify the user interface to the application.

modified data object, and repeat. See [12] for examples of the types of workflows enabled by the Gemstone frontend.

As stated above, the Gemstone frontend is built on top of the Mozilla source code. We use a new suite called XULRunner [17] which is a full runtime environment composed of the same libraries used to build the Firefox web browser and the Thunderbird mail client. As with all Mozilla-based applications interface elements are defined with XUL and rendered by the Gecko runtime engine; the logic driving the interface is defined through XPCOM components and JavaScript. The Gemstone frontend supports three major platforms: Microsoft Windows XP, Linux, and MacOSX. It uses Mozilla's mechanisms to detect new versions of the framework and will prompt the user before updating to the newest version.

3.1 Registry and Discovery of Services

Application web services have endpoints and interfaces that need to be known to the PSE before they can be made available to the end-user. Within the web services toolset, the Web Service Description Language (WSDL) [14] is used to describe the service interfaces and the parameters. For registration and discovery, there are a number of existing and emerging standards including the Uniform Description Discovery and Integration (UDDI) specification [30], Web Services Inspection Language (WS-Inspection) [24], and WS-Discovery. UDDI is fairly heavy-weight and quite complex requiring additional server systems to be setup and maintained. While WS-Inspection and WS-Discovery address some of UDDI's shortcomings and may be of use in the future, neither specification has yet been accepted by any standards body nor seen widespread adoption in the community.

The Gemstone approach to service registration and discovery utilizes a much simpler decentralized mechanism. The registry is simply a file made available by a standard web server at a specific URL. The file format is in RDF [6] and it can be easily incorporated into virtually any client system. In addition to the Gemstone frontend, this discovery mechanism has been incorporated into python-based PSEs and Java-based workflow systems. Because of the simplicity of the registration and discovery infrastructure, organizations can easily publish the set of web services available to their users.

For example, the registry provided by the NBCR services infrastructure is available at: `http://rocks-106.sdsc.edu:8080/xul/registry.rdf` and contains the following structure:

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:s="http://www.sdsc.edu/rdf/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

<!-- first define the categories -->
<rdf:Bag rdf:about="http://www.sdsc.edu/rdf/services">
  <rdf:li>
    <rdf:Seq rdf:about="http://www.sdsc.edu/rdf/chemistry"
      s:category="Chemistry">
      <rdf:li rdf:resource="http://www.sdsc.edu/rdf/chemistry/0"/>
      <rdf:li rdf:resource="http://www.sdsc.edu/rdf/chemistry/1"/>
      <rdf:li rdf:resource="http://www.sdsc.edu/rdf/chemistry/2"/>
    </rdf:Seq>
  </rdf:li>

<!-- add the other categories  ... -->
</rdf:Bag>

<!-- now the services  ... -->
<s:service rdf:about="http://www.sdsc.edu/rdf/chemistry/0"
  s:name="Apbs"
  s:url="http://rocks-106.sdsc.edu:8080/xul/apbsMain.xul"
  s:wsdl="http://rocks-106.sdsc.edu:8080 \
         /axis/services/APBSBlockingPort?wsdl"/>

<!-- a secure service (requires authentication)  ... -->
<s:service rdf:about="http://www.sdsc.edu/rdf/matsci/0"
  s:name="Siesta"
  s:url="https://rocks-106.sdsc.edu:8443/xul/siestaMain.xul"
  s:security="true"
  s:wsdl="https://rocks-106.sdsc.edu:8443 \
/axis/services/SiestaServicePort?wsdl"/>
<!-- etc ... -->
</rdf:RDF>
```

The first portion of the registry defines the broad categories of the services, Chemistry, Biology, Materials Science, etc. The second part defines each individual service: the name, the URL of the service panel definition, an optional flag to indicating that authentication is required, and finally the URL location of the WSDL defining the service. For services requiring authentication, the service panel can support the display and acceptance of an end-user license. New registry entries may also include custom tags with additional information for the service.

The Gemstone frontend provides the user with a familiar URL bar (see Figure 3), similar to what is provided by most web browsers, allowing the user to effectively browse the services offered by various service providers. We are currently working with various application providers to ensure that the registration capabilities can meet their needs.

3.2 Security

The Gemstone frontend integrates the Grid Security Infrastructure (GSI) [21] with the native security mechanisms provided by the Mozilla framework and supports authenticated access to and access control for the remote application web services. As with Topaz, Gemstone leverages the GAMA system to provide user authentication services and supports authorization-based access to the remote application web services.

Operationally, the user specifies a host for a GAMA server in the Gemstone frontend preferences. Then the user supplies the user and password for his or her certificate to the right of the registry URL bar and pushes the login button. Gemstone contacts the GAMA server over a secure SSL connection and supplies the username and password for the user. The GAMA server authenticates the user and returns a PKCS12 certificate which is loaded into the local security repository provided by the Mozilla core. At this time, Mozilla does not support x509 proxy certificates which are short term self-signed certificates, hence we use the actual certificate in the PKCS12 format. To protect the user's private key, the local certificate repository is itself protected by a Master Password and we ensure that the certificate is deleted properly after use.

Once the certificate is loaded the use of secure services becomes transparent to the user and requires no further intervention on their part. Mutual authentication is performed during the SSL handshake before the user's request can proceed. For the remote application web service implementation, the service invocation request passes through a special Grid-Map Authorization Handler where access control mechanisms to authorize the user are applied. If the process fails at any point, an error message is returned to the user detailing the reason. If the server supports encryption then all web services communication is encrypted at with the strongest SSL cipher suite shared by Gemstone and the server.

3.3 Service Panels

As described above, the service panels are the main interaction between the user and the remote application services. The applications are services in the Web Service sense, with an interface definition describing the various methods and parameters for those methods. The service panel contains all the user interface elements to set various application parameters, attach data files, start the computation and retrieve output files.

A key capability in the Gemstone architecture is that the application developers – those who create the application service interface – are the ones who define the user interface. Traditionally the application developer may develop the service API, but the user interface is left for the client application developer who may not have the knowledge or experience with the application to understand how the various parameters are specified, the inter-dependencies between the parameters, or the best presentation of the parameters.

For example, consider the quantum chemistry application GAMESS that has over 300 different "control groups" with each control group providing a number of different parameter options. Many control groups have dependencies: the "PDC" control group implies that control group "ELPOT" must be defined. The existence of other control groups and parameters negates or overrides other control groups in complex ways. Building an user interface for such an application needs to be done with significant application developer assistance. With the Gemstone infrastructure the developer can provide the entire user interface and continue to evolve it with the clients dynamically integrating the newest versions as they are deployed.

The service panels are described with XUL. Business logic – what happens when buttons are pressed, for instance – is defined in JavaScript. The XUL and JavaScript portions can be developed independently and updates to either are immediately available to users.

When a user selects a service panel in the service list, the Gemstone frontend downloads the corresponding XUL and JavaScript files and integrates them into the framework. The XUL file defining the service panel is loaded into a new tab in the main content area. The code in the JavaScript is also loaded, but within a security context that restricts its access to local resources. Gemstone framework APIs are added into the execution scope of the running panel to give access to user security features, the local filesystem, inter-panel communication, and the Job Manager. As Gemstone develops, additional APIs will be added as needed.

Gemstone's Job Manager APIs are used to launch and manage remote service invocations on behalf of the user. When a service panel is ready to initiate a invocation – say for instance when the user selects the "Run" button in the service panel – the service panel creates a new Gemstone Job object and sends it to the Job Manager to begin execution. Job information, including input parameters if requested, is written to local storage and the web service is invoked. The Manager will poll the web service for status at a modifiable interval, updating both local storage and the Job Manager Viewer with real-time information. It is typical that a remote service invocation may result in a long running job or set of jobs being scheduled for execution on a computational cluster. As these jobs may take significant time to complete, the user may close the service panel or the Gemstone application at any time; when he or she returns to check progress a new service panel will be created containing their pre-filled input options and current status. For applications like GAMESS, the service panel will also refresh graphs if necessary. For the service panel developer, the Job Manager APIs make it simple to interact with the running job; each panel is automatically sent notice at each stage of its job's lifecycle and may choose to ignore or further process the corresponding web service response.

Service panels typically act autonomously but there is often a need to share information with the framework or communicate with other panels. The Gemstone frontend supports a publish-subscribe communication model for one-to-many notification. A panel can register interest in a particular event or send event notifications through a global observer service which is built into the framework. A typical example is the "login-complete" event generated after a successful login. A service panel can use this information to allow different "secure" options or display the user's DN which is sent as well.

Aside from the security constraints, the developer of the service panel can build as simple or as complex interactions as they wish. The XUL technology is simple enough to enable non-programmers to develop basic functionality such as text boxes and buttons. The following is the XUL fragment associated with the PSize service panel and shows that XUL development is in many ways simpler to HTML forms development. More sophisticated interactions are also possible using advanced JavaScript techniques such as AJAX, Dynamic HTML, even Macromedia's Flash – the same techniques that are used for highly dynamic Web 2.0 sites.

```
<vbox id="main-frame" style="overflow: auto"
observes="drag" flex="1">
  <groupbox id="input-box" observes="drag">
  <caption label="Input" />
   <grid>
    <rows>
     <row />
     <row />
    </rows>
    <columns>
     <column >
      <button label="Run" oncommand="runPsize(  )"
id="run-btn" observes="drag"/>
<label value="Progress:" />
        <label value="Status Message:" />
    </column>
   </grid>
  </groupbox>
  <groupbox id="output" flex="1">
   <caption label="Output" />
     <textbox id="output-txt" value="" multiline="true"
flex="1" rows="40" minheight="350"/>
  </groupbox>
</vbox>
```

Deployment of a service panel is as simple as locating the corresponding XUL and JavaScript file on a web-accessible location, typically hosted at the same location of the application service deployment.

3.4 Visualization

The Gemstone frontend also provides visualization capabilities. Figure 4 shows two examples. The first is integrated into the GAMESS service panel. As an energy minimization calculation is running on the remote resource, the output service panel shows the GMAX and GRMS outputs – the total energy of the molecular conformation and the gradient of the molecular energies vs the number of steps. This particular runs shows the user that the calculation is not converging properly due to incorrect input parameters. Currently this capability is built directly into the service panel, but we are working to move this kind of graphing capability directly into the framework so that it can be leveraged by any service panel in a standard way.

In addition, Figure 4 also shows molecular visualization of molecules integrated into the Gemstone frontend by the Garnet component. Garnet uses JOGL (Java for OpenGL) to display three-dimensional visualizations of even very large molecules and uses the capabilities of accelerated graphics cards where available. Garnet is an alternative to other molecular viewers as it offers the ease of deployment that a web-based applet would yet all the advantages of a desktop viewer. Using the communication method described earlier Garnet can interact with the service panels as needed.

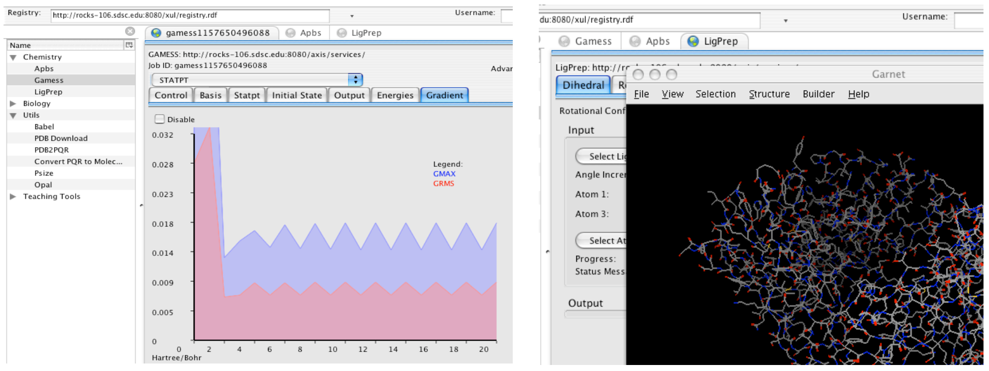


Figure 4. Two types of visualization capabilities.

4 Related Work

One of the most prevalent mechanisms for building PSEs for major grid computing projects is web-based portals that provide end-users equipped with a web browser access to a heterogeneous set of tools and services. Just as with Topaz and Gemstone, these web portals provide data management capabilities (in many cases providing a front-end to the GridFTP servers), single sign-on security, and access to computational applications and services. Many toolkits exist for building web or grid portals, both open source [28, 33, 22] and commercial.

One of the main advantages of a web-based portals over our approach is that they support collaboration and data sharing across users. For example, the Geon Portal [27] serves as a repository for datasets relating to geoscience and has over 400 registered datasources submitted by their users. Other users can search the repository for relavent datasets using metadata and ontologies and integrate the datasets for their work. Because they provide a central site that all users go to for accessing data, the server can maintain state in order to share data and knowledge across the user community.

However, any central web portal site requires significant maintenance and administration for operation. In addition, it requires significant development effort to add additional applications or services. For example, to integrate a new application would require web development on the portal, hence this model would not be appropriate in an environment where applications or resources are being added or removed frequently. The Web Services Remote Portlet (WSRP) [29] specification may evolve to address this specific capability by allowing the user interface elements of a portal to be defined at a remote site; however, the specification and implementation are still quite limited.

Secondly, while web-based tools support ubiquitous access through the browser, the are inherently limited: they can not access local resources on the user's desktop, and are mostly limited to HTML which is not very interactive. New AJAX frameworks may help here, but because the Gemstone frontend is built using the same technology, any advances in web development can be leveraged in our framework in addition to better access to the user's desktop.

Also building on the .NET framework and very similar in concept to the Gemstone frontend is the Notebook project [32] that is "client side data repository, collaboration environment and smart client for SOAP-based web services." While the capabilities are similar, the technologies used to build the Notebook are specific to the next-generation of Windows, specifically XML Application Markup Language (XAML).

The Java Cog [35] also integrates grid protocols into a set of Java libraries that can be used to build rich client or desktop applications that can access grid resources. In particular, the Java Cog supports the services of the Globus Toolkit including GridFTP.

For data management, web-based access through portals means that file uploads must first be staged on the web server before being copied to the GridFTP server. Using Topaz, the user can directly access the remote data repository securely using established grid security protocols without having to install grid libraries or additional tools.

Other groups have also been developing similar capabilities: MyGridFTP [31] and WSRF.NET [20] provides similar capabilities by building custom client applications using the Windows .NET framework.

5 Summary

In this paper we describe an approach to building an extensive PSE based on the Mozilla framework that provides users with access to remote grid resources including data and services. Our approach integrates grid protocols directly into desktop applications enabling a high level of interactivity and seamless integration with both desktop resources and remote grid resources.

Topaz, the data management component, extends the capabilities of the Firefox browser to support the GridFTP protocol. User's simply specify a gsiftp URL and the extension does the rest. There is no requirement for Globus to be installed on the user's machine, hence access to data managed by the GridFTP servers is greatly simplified. The current version of Topaz supports both data download and uploads; support for third-party-transfer is in development. Since we integrate the Globus implementation of the GridFTP client libraries into our extension, the performance is what you would expect – very little additional overhead is added by Topaz over and above the performance of the native Globus tools.

A beta release of the Topaz extension is currently available [8] and supports MacOSX and Linux.

Gemstone provides a complete end-user environment for the discovery of and access to remote application services. Because the registries used by Gemstone are defined by a simple RDF file published on a website, it is very simple for institutions to publish and maintain their list of services. Gemstone allows users to *browse* these services and provides a framework to integrate those capabilities dynamically, while allowing the application provider to control the functionality of that interface. Gemstone leverages the current state-of-the-art in web technologies, including support for JavaScript and AJAX, SVG, Canvas, and CSS.

Version 1.0 of the Gemstone frontend has recently been released and is available for MacOSX, Linux and Windows [2]. It provides access to the application services developed by the National Biomedical Computation Resource (NBCR) and includes applications in the Biomedical and Computational Chemistry domain: AutoDock, APBS, GAMESS, Siesta, LigPrep, Protein Data Bank, and many associated utilities.

6 Acknowledgments

We thank the NSF for supporting the Gemstone and Topaz projects through the Middleware Grant SCI-0438430. Topaz was also supported by NSF EIA-0080940, Graduate Education for Minority Students in Computer Science and Engineering, and NSF SCI-0506429, Dynamically Adaptive Protein-Ligand Docking System based on multiscale modeling. We also would like to thank the NIH for supporting NBCR through the National Center for Research Resources program grant P41RR08605 that supports the development of the computational infrastructure.

References

[1] Biomedical Informatics Research Network (BIRN). http://www.nbirn.net/.

[2] The Gemstone Project. http://grid-devel.sdsc.edu/gemstone/.

[3] The Mozilla Add-ons. http://addons.mozilla.org/.

[4] The Python Molecular Viewer (PMV). http://www.scripps.edu/ sanner/python/pmv/.

[5] The Vision programming environment. http://www.scripps.edu/ sanner/python/vision/.

[6] RDF/XML Syntax Specification. Technical report, W3C, Feb 2004. http://www.w3.org/TR/rdf-syntax-grammar/.

[7] Teragrid Science Gateways. online documentation, Sept 2006. http://www.teragrid.org/programs/sci%5fgateways/.

[8] The Topaz Firefox Extension, 2006. http://gcl.utep.edu/projects/topaz/.

[9] W. Allcock, J. Bester, J. Breshnahan, A. Chervenak, L. Liming, and S. Tuecke. GridFTP: Protocol Extensions to FTP for the Grid. Internet Draft, March 2001. http://www.gridforum.org.

[10] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludaescher, and S. Mock. Kepler: An Extensible System for Design and Execution of Scientific Workflows. In *16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, 2004.

[11] K. Bhatia, S. Chandra, and K. Mueller. GAMA: Grid Account Management Architecture. In *IEEE International Conference on EScience and Grid Computing*, 2005.

[12] Karan Bhatia, Stephen Mock, Sriram Krishnan, Jerry Greenberg, Brent Stearn, Kim Baldridge, and Celine Amoreira. Grid Workflow Challenges in Computational Chemistry. Technical Report SDSC TR-2006-7, San Diego Supercomputer Center, 2006.

[13] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, and B. Moe. Wide Area Data Replication for Scientific Collaborations. In *6th IEEE/ACM Int'l Workshop on Grid Computing (Grid2005)*, Nov 2005.

[14] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL). Technical report, W3C, 2001. http://www.w3.org/TR/wsdl.

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

[16] The Mozilla Corporation. XPCOM. http://www.mozilla.org/projects/xpcom/.

[17] The Mozilla Corporation. XULRunner. http://developer.mozilla.org/en/docs/XULRunner.

[18] K. Czajkowski et al. WS-Resource Framework, May 2004. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.

[19] T. Oinn et al. Taverna: A tool for the composition and enactment of bioinformatics workflows. In *Bioinformatics Journal*, volume 20(17), 2004.

[20] J. Feng, L. Cui, G. Wasson, and M. Humphrey. Toward Seamless Grid Data Access: Design and Implementation of GridFTP on .NET. In *Proceedings of the 2005 Grid Workshop*, 2005.

[21] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *ACM Conference on Computers and Security*, 1998.

[22] The Sakai Foundation. Sakai: A Collaboration and Learning Environment for Education, 2006. http://www.sakaiproject.org/.

[23] Stratis Gallopoulos, Elias Houstis, and John Rice. Computer as Thinker/Doer: Problem-Solving Environments for Computational Science. *IEEE Computational Science and Engineering*, 1994.

[24] IBM, Microsoft. Web Services Inspection Language (WS-Inspection). http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html?dwzone=webservices.

[25] Sriram Krishnan, Kim Baldridge, Jerry Greenberg, Brent Stearn, and Karan Bhatia. An End-to-end Web Services-based Infrastructure for Biomedical Applications. In *6th IEEE/ACM International Workshop on Grid Computing*, 2005.

[26] J.L. Moreland, A.Gramada, O.V. Buzko, Qing Zhang, and P.E. Bourne. The Molecular Biology Toolkit (MBT): A Modular Platform for Developing Molecular Visualization Applications. *BMC Bioinformatics*, 6(21), 2005.

[27] Ullas Nambia, Bertram Ludaescher, Kai Lin, and Chaitan Baru. The GEON Portal: Accelerating Knowledge Discovery in the Geosciences. In *Proceedings of the 8th ACM International Workshop on Web Information and Data Management (WIDM 2006)*, Nov 2006.

[28] Jason Novotny, Michael Russell, and Oliver Wehrens. Gridsphere: An advanced portal framework. *euromicro*, 00:412–419, 2004.

[29] OASIS. WSRP 1.0 specification. Technical report, OASIS, Aug 2003. http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf.

[30] OASIS. UDDI Version 3.0.2. Technical Standard, Feb 2005. http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm.

[31] A. Paventhan and K. Takeda. MyGridFTP: a zero-deployment GridFTP client using the .NET framework. In *Advances in Grid Computing (EGC 2005) European Grid Conference*, Amsterdam, The Netherlands, Feb 2005.

[32] G. Quinn, B. Jennings, and M. Miller. The Notebook Project: Connecting Research Environments, 2006. http://www.notebookproject.org/.

[33] Charles Severance. Integrating Grid Capabilities into the CHEF Collaborative Portal Framework. Technical Report NEESgrid-2003-01, NEESGrid, 2003.

[34] S. Subramaniam. The Biology Workbench: A Seamless Database and Analysis Environment for the Biologist. *Proteins: Structure, Function, and Genetics*, 32(1):1–2, 1998.

[35] G. von Laszewski, J. Gawor, S. Krishnan, and K. Jackson. *Grid Computing: Making the Global Infrastructure a Reality*, chapter 25, Commodity Grid Kits - Middleware for Building Grid Computing Environments. Wiley, 2003.

[36] Richard Zamudio, Daniel Catarino, Michela Taufer, Brent Stearn, and Karan Bhatia. Topaz: A Firefox Protocol Extension for GridFTP based on Data Flow Diagrams. Technical report, University of Texas at El Paso, April 2006.