VisPortal: Deploying grid-enabled visualization tools through a web-portal interface

Wes Bethel* ewbethel@lbl.gov
Cristina Siegerist* cesiegerist@lbl.gov
John Shalf* jshalf@lbl.gov
Praveenkumar Shetty* psshetty@lbl.gov
T.J. Jankun-Kelly*† tjk@acm.org
Oliver Kreylos*† kreylos@cs.ucdavis.edu
Kwan Liu Ma† ma@cs.ucdavis.edu

Abstract

The LBNL/NERSC Visportal effort explores ways to deliver advanced Remote/Distributed Visualization (RDV) capabilities through a Grid-enabled web-portal interface. The effort focuses on latency tolerant distributed visualization algorithms, GUI designs that are more appropriate for the capabilities of web interfaces, and refactoring parallel-distributed applications to work in a N-tiered component deployment strategy. Most importantly, our aim is to leverage commercially-supported technology as much as possible in order to create a deployable, supportable, and hence viable platform for delivering grid-based visualization services to collaboratory users.

1 Introduction

Visualization systems will become an essential part of the emerging fabric of Grid services. Nascent Science Grid and collaboratory development efforts are pushing a new paradigm of remote HPC resource usage where scientists who used to login to the supercomputer of their choice to submit computing jobs are now presented with a web/portal interface that can assign their computational workload to a pool of resources anywhere in their Virtual Organization (VO). This can lead to a very complicated environment in which to perform data analysis when the data can end up spread over countless resources in the virtual organization. The LBNL/NERSC VisPortal effort explores ways we can deliver grid-based advanced visualization and data analysis capabilities through this same web/portal interface paradigm in order to unify data analysis with the rest of the portal-based collaboratory environment. The resulting work should be suitable for embedding or interoperation with other web/portal mediated collaboratory efforts.

The grid offers the abstraction of transparent access to distributed computing resources, among them computing cycles, storage, software and licenses, and special equipment (instruments). However it is still a complicated task to use grid-enabling software. Despite significant recent advances in grid software packaging, installation of grid services on client machines is still very difficult for novice users. Even a simple standalone grid-enabled application requires the developers and users to navigate a minefield of configuration, application and library dependencies. Hiding the software complexity behind a grid-enabled portal alleviates many of these deployment problems by bringing the resources together through a much simpler user interface that requires little or no additional installation on the client side. The portal aggregates the distributed resources via a single point of presence, allowing the scientist to access them from an easy to use interface.

In our case we want to provide NERSC (National Energy Research Scientific Computing Center) users with a centralized point to access their data, computing and visualization resources available in our center. NERSC users are distributed nationwide and access NERSC computers from a variety of platforms. The portal paradigm offers a means to deliver customized visualization and data management services to users with less of a burden on porting complex software packages to the heterogeneous machines and software environments employed by our user community.

*Visualization Group, Lawrence Berkeley National Laboratory, Berkeley, CA 94720

†Visualization and Graphics Research Group, Center for Image Processing and Integrated Computing, Department of Computer Science, University of California, Davis, CA 95616

2 VisPortal

The LBNL/NERSC VisPortal effort explores ways to deliver Grid-based advanced visualization and data analysis capabilities through a Web portal interface. Using standard Globus-grid middleware and off-the-shelf web automation, the VisPortal hides the underlying complexity of resource selection and distributed application management on a sea of heterogeneous distributed computation and storage resources. The portal automates complex workflows like the distributed generation of MPEG movies or scheduling of file transfers, mediates access to limited hardware resources like our visualization server, Escher's, offscreen hardware graphics pipes, and controls the launching of complex multicomponent distributed visualization applications like Visapult – an application used for remote and distributed, high performance interactive volume rendering of massive remotely-located datasets.

Another form of interaction the portal provides is a DHTML spreadsheet-like environment. The web-based DHTML interface provides a ubiquitously accessible UI to the users that encapsulates the visualization and data exploration process. From a single access point, the user can browse through the data (probably where it was generated), launch all components of a distributed application with all of the complex commandline arguments and configuration information necessary to knit them together into a single functioning application. This considerably lowers the bar for client side complexity.

While portals offer considerable advantages over standalone grid applications in terms of managing client-side complexity, working within the portal paradigm has not been an easy task. While it shields the users from the underlying complexity of the grid, the same cannot be said for developers who are attempting to use the portal paradigm. We will discuss the many problems that have yet to be solved before the portal can become a dependable production resource. The excercise of creating a working implementation with a very controlled set of users is critically important for identifying the deficiencies in current technology and driving the direction of future middleware development.

3 Architecture

This portal builds upon the Grid Portal Development Toolkit (GPDK) developed by Jason Novotny [11]. GPDK makes use of the Tomcat/JSP 3.x web engine and the Java CoG 1.0a [15] to provide the JSP engine direct access to Grid services. JSP is extremely popular for e-commerce applications so we have many opportunities for synergy with commercial web automation environments. Also, based on the experiences of several previous portal implementation efforts, JSP is considerably more readable and maintainable than Perl-CGI or Python-CGI methods. [3] [16] [2]

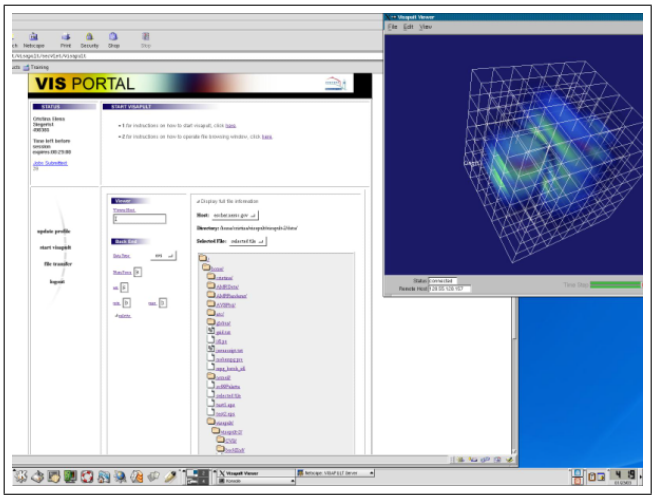

Figure 1. The Visportal Launching the Visapult distributed parallel volume rendering application.

4 Services

The visualization portal has been a proving ground for a number of different delivery paradigms. This includes thin-clients, slender-clients, and thick clients. The thin-client interface means the entire GUI is presented in DHTML backed by the server-side JSP engine. The slender-clients involve very thin GUIs such as Java applets lauched by the web-browser on the client machine, that provide a front-end for massively parallel or distributed/multi-tier visualization back-ends like Visapult or offscreen rendering pipe access. Finally, the thick clients simply use the portal as a broker for locating remote data or services that extend the capabilities of a standalone tool like OpenDX or AVS Express. We are using the portal as a testbed for testing out these various application deployment design patterns and compare the relative merits of their implementations in terms of development complexity, deployment complexity, and client-side user experience.

These services include:

- *File Management Interface:* This is a fully DHTML/thin-client interface for managing imme-

diate mode transfers.

- *MPEG Movie Generator:* This is perhaps the most-requested service for many visualization groups. Albeit, the back-end of this tool can be operated entirely from the commandline, we are investigating whether GUI interfaces to commonly used commandline tools offers some advantages for our collaboratory/portal users.
- *AMR Volume Renderer:* This tool explores ways to expose visualization hardware and software (hardware graphics accelerators or special-purpose applications) as grid services. This service is an interface to a hardware-accelerated volume renderer for hierarchical adaptive meshes that makes use of SGI Infinite-Reality Engine graphics pipes.[8]
- *AMR Websheets:* The websheets [6] interface uses the same back-end as the volume render, but investigates an alternative visualization spreadsheets GUI paradigm for drilling down into complex multidimensional parameter searches.
- *Visapult:* This multicomponent distributed application uses image-based rendering methods employed by Visapult are able to hide much of the latency of the intervening network [13]. Manually launching these distributed components proves to be very tedious without the VisPortal's automation.

4.1 File Transfer

The interface for 3rd party file transfers is perhaps the most mundane of the services offered by the portal, aside from the login screen. However, it has the potential to be the most broadly used of the services. It provides portal users with full access to GUI for 3rd party file transfers and remote file browsing without having to install any Globus/Grid software locally on their machine. It also makes the file transfers considerably simpler and less error prone than attempting the same feat using very long file paths in URL syntax on the commandline.

While based on the original GPDK back-end implementation, the user interface was completely rewritten to sport a more-"Windows"-like directory interface with graphical icons that differentiate directories and various file types. The GUI is still drawn using lowest-commond-denominator DHTML so that it works in a wide variety of web-browsers without any special client-side considerations.

A more important contribution of this interface is that it has been embedded in virtually all of the other VisPortal interfaces as the primary method for browsing remote filesystems to select datasets for the visualization tools that the portal manages. It would be nice to simply package that graphical element as a "component" that could be dropped in to any of the VisPortal applications, but the limitations of JSP syntax make it impossible to treat this graphical element the way one would a "widget" in a desktop GUI interface. Web automation interfaces based on the Portlet/OGSA technologies promise to make such graphical components possible – validating some of the key reasons for moving to that technology.

Another deficiency of the initial FTP interface implementation is that it only supports immediate mode file transfers. Theoretically, a user would be motivated to use the portal's file transfer interface because the files they are managing are far too large to move to their own workstation. A user interface that supports only synchronous file transfers offers little to benefit in that situation. An updated version of the portal's implementation of the FTP interface that supports queueing of background transfers will be released soon. We are also working with Shreyas Cholia on his java-applet-based "Globus File Yanker" [1] that can schedule offline, reliable 3rd-party transfers between HPSS systems and distributed disk caches.

4.2 MPEG Generator

One of the portal's potential usage paradigms is a management interface to existing desktop software packages or simple forms interface that controls batch-mode data analysis. For instance, the portal could provide a remote interface to commonly used scripts for batch-mode post-processing of datasets. All of the processing is initiated by a simple HTML-forms interface and is performed entirely by back-end hosts that are controlled by the portal using the Java CoG.

An example of a basic service is an MPEG movie generator that invokes IDL (Research System Inc's Interactive Data Language[5]) on our visualization server. The user selects a data file in one of his/her grid-enabled resources and, after selecting MPEG generation parameters in the portal interface, launches the IDL job. Third party file transfer is used to transfer the data file to the visualization server where an IDL script can run in batch mode – launched by a GRAM. When the job completes an email tells the user how to access the MPEG file. The portal simplifies the manual staging of components. Without the portal, this user would need to transfer the data files from an IBM SP to the visualization server, log in this machine, run the IDL script with appropriate parameters, and finally

transfer the MPEG back to his own resources. With the portal, the entire process is initiated by selecting the image files and pressing the *start* button and the grid middleware enables a single portal host to expand its computational capability by farming the tasks off to a pool of remote resources. Any visualization application (IDL, Express, VTK) that allows batch processing could be managed by the portal in this manner.

4.3 AMR Volume Renderer

The AMR volume rendering service takes advantage of graphics hardware to accelerate the offscreen volume rendering of hierarchical adaptive meshes created by Adaptive Mesh Refinement (AMR) [10] simulation codes.[8] The client component of this application presents a simple user interface and an image window that looks for all practical purposes like the entire application is running locally. The back-end of the application, however, can make use of the offscreen hardware rendering using SGI Onyx IR2 pipes or a parallel software implementation that can run in a cluster environment. The application can adjust dynamically to changing resource and network conditions by varying the level of JPEG compression as well as the depth it descends in the AMR hierarchy for volume rendering. The user interface remains the same despite entirely different back-end implementations and deployments. Other tools in our portal, such as the AMR WebSheet, share this very same volume rendering service as their primary back-end computing engine.

The portal's primary role is to select an appropriate resource to run the back-end, launch the client-side GUI to the application with appropriate parameters to automatically connect to the back-end. We envision an even more integrated role for the portal in mediating resource conflicts. Currently, the resource scheduler and access-control mechanisms operate on a very rudimentary first-come-first-serve basis. If the graphic pipe on the Onyx is already in use, then further attempt to launch the AMR renderer there simply fail. A more advanced system will mediate access to such limited resources by either time-slicing the access or by working in concert with a scheduling and reservation system that allows dedicated use of the resource via a calendar interface.

We also see a possible role for the portal in mediating shared viewing of the graphics framebuffer contents for the volume rendering application for collaborative applications. Currently, only one user is authorized to connect to the image-delivery channel of the application at a time. In a collaborative scenario, the portal would manage authorization or distribution of the current framebuffer contents to multiple viewers in conjunction with synchronous collaborative services such as the Access Grid.

4.4 AMR WebSheet

The WebSheet interface [6] uses the same back-end as the volume render, but investigates an alternative "visualization spreadsheet" GUI paradigm for drilling down into complex multidimensional parameter searches. The interface structures the visualization process by providing an intuitive display of the visualization parameter space—the spreadsheet is a two dimensional window into this space (Figure 2). At any time, only two parameter types are displayed along the row and columns—the default values for the other parameters are displayed along the top. Thus, a user always has context for where they are in the exploration. In addition, the tabular structure of the interface allows for quick comparison of previous results, providing context for where a user has been. Interaction with the interface occurs by adding or removing new rows or columns, changing which parameters are along the rows or columns, and changing the default parameter values. When a cell is rendered, the default parameters values are combined with parameters from the cell's row and column in order to generate a result. The web-interface is implemented entirely in JavaScript and standard HTML.

The WebSheet makes use of a separate application server than the main VisPortal. This application server captures the user's visualization process. By capturing the process, the system ensures that the visualization results generated, and the relationships between those results, are not lost when the visualization session ends. To record the visualization process, a formal model of the visualization process is used [7]. Requests for an image from the interface passes through the web application server; these requests are then forwarded to the back-end renderer. As each requested image is rendered, the corresponding visualization session result is stored by the web application server. Thus, at the end of a session, all the rendered images, the parameters used for creating that image, when that image was generated, and that image's relation to previous images are available for later use.

The WebSheet and its application server were designed with collaboration in mind. As mentioned, the user's exploration process is stored automatically by the application server. At a later time, a collaborator can reload this session into another WebSheet to extend the exploration. In addition, the application server can display an HTML overview of the entire session (Figure

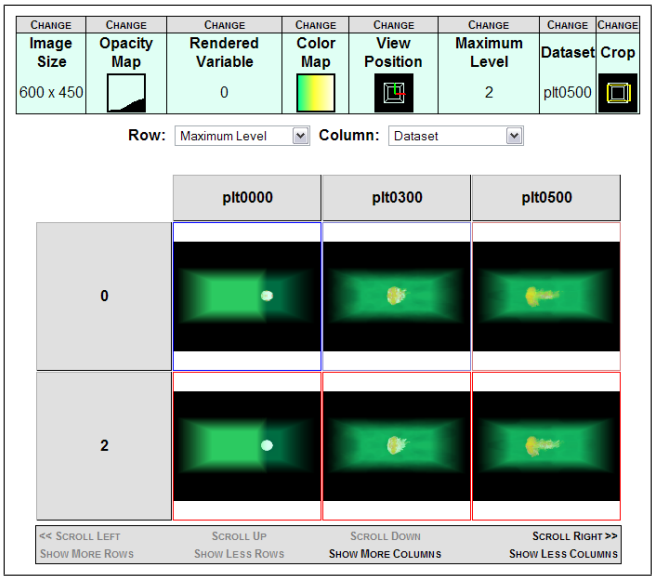

Figure 2. The AMRWebSheet interface, an example of the web interface to grid-based visualizations. The interface consists of three major areas: The default parameter bar that displays and allows the modification of the default parameter values; the displayed row and column parameter drop-down lists; and the tabular result display. The first two components are used to change the location of the tabular window in visualization parameter space while the last component is used to request the rendering of new results.

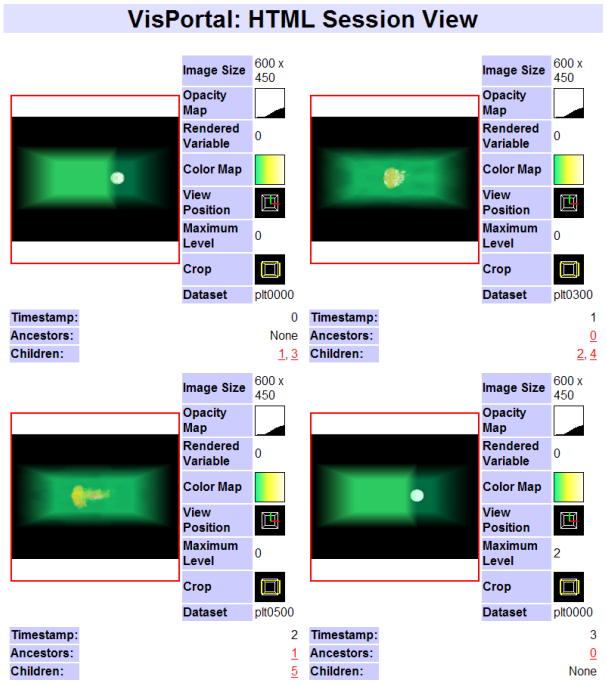

Figure 3. HTML session page for the session in Figure 2. The page provides a summary of the visualization session and supports the annotation of results.

3). This overview serves two purposes. First, scientists can share this web page with collaborators to easily. Secondly, the collaborators can gain an understanding of the process and its results quickly without having to use the full web interface. It is envisioned that scientists could annotate the HTML overview page in order to share the knowledge gained using the VisPortal.

The web application servlets that manage visualization sessions are implemented in Python using the Webware web application environment (http://webware.sourceforge.net/). A group of servlets create, process, and store sessions. When a the VisPortal JSP engine forwards a connection to the Webware server, a new session—identified by a temporary cookie—is created in addition to servlet-persistent objects. Whenever a user interacts with the generated HTML interface—the AMRWebSheet—HTTP requests are communicated to the interface servlet indicating that the behavior fired. This request in turn modifies the visualization session state. When the client needs to be updated—e.g., after result generation—a server-initiated refresh is performed to display the new information. Finally, when a session terminates or expires due to inactivity, the session results are encoded as an XML document on the web application server for later retrieval as described previously.

4.5 Visapult

Visapult (Figure 1) is composed of 3-distributed components, a slender-client viewer that connects to a massively parallel remotely located volumer-rendering back-end which in turn connects to a parallel data source like a running simulation code (like Cactus) or a distributed disk-cache like DPSS. The image-based rendering methods employed by Visapult are able to hide much of the latency of the intervening network [13]. Visapults highly tuned network implementation has enabled it to win the annual SC SCinet Bandwidth Challenge Competition three years in a row.

Visapult was the very first visualization applica-

tion hosted by the VisPortal system. Wes Bethel, who originally developed Visapult for the NGI Combustion Corridor project, discovered very quickly that it would be virtually impossible to hand out his new multicomponent, distributed, parallel volume rendering application to a novice user. While the distributed architecture of Visapult offered huge performance advantages for scientists who needed to study very large remotely-located datasets, launching the tool involved very tedious commandline arguments and a particular order for starting up each of the distributed components that comprised the application. For that matter, unless the user had already performed an multi-way exchange of ssh public-keys, they would have to log in to no less than 3 different machines to prepare the components for launching. The GSI security model addresses the problem of impractical ssh key exchanges needed to securely launch these distributed visualization applications. The portal offers considerable advantages in automating the launching process so that a user merely selects a file displayed in their web-browser and presses the start-button to launch all of Visapult's distributed components. Such applications would remain entirely impractical to use were it not for some automated launching process that hides the complexity of heterogeneous pipelined distributed applications that are becoming increasingly important for analysis of large remotely located datasets.

4.6 Collaboration

Aside the AMR websheet's ability to share session state between users, the portal is primarily used as a platform to explore different paradigms for exposing distributed visualization methods as grid services. Collaboration, however, isn't itself a service – it is perhaps better described as a workflow. Web logic, with its hyperlinks and seemlingly location-independent access to distributed resources, offers an excellent platform for representing workflows. We have observed repeatedly that people who do not actively collaborate with one another without the assistance of advanced collaborative interfaces, are unlikely to begin collaborating by virtue of such an interface. Therefore, we regard the task of creating a collaborative interface as being intimately tied to creating custom portal application interfaces that support existing collaborative workflows established by various research communities rather than a generic *service* that we offer to the community. The various pieces of this portal will be used to rapidly implement workflows for specific applications, like automated creation of movies from completed Global Climate Modelling jobs that can be posted on the web and shared among members of the climate modelling community. Such a service would complement existing shared data repositories established by the climate modeling community.[4]

One significant service required to support collaboration is the ability to share data with collaborators of your choosing. Another role for the portal is management of information that a given community of researchers wants to share in this manner. At a coarse level, the portal acts as a is a repository management system or central index for shared data files. At a finer grained level, the portal can manage annotations that are embedded into collaboratively generated datasets. This can be a part of the integration with online/synchronous collaborative systems like the AG, but we consider persistent management of annotations that result from offline collaboration to be a critical requirement. In the case of the AMR WebSheets, the state of the websheet can be stored on the portal and shared with other portal users by implementing our own permissions mechanism within the portal framework. The portal programmer has complete control of security on the machine that runs the portal automation software and can implement a wide variety of security policies. However, larger datasets, that cannot possibly fit on the portal cannot be shared as easily due to the lack of support for fine-grained in currently deployed grid infrastructure.

Currently, access control policy is enforced by mapping a user's distinguished name to a particular user account. This makes the groups of users subject to the file access permissions and access control policies of a particular host or operating system – making ad-hoc sharing of data very difficult. The CAS (Community Authorization Service) circumvents this issue by offering access to shared/group credentials, but runs afoul of organizational policies that do not permit group accounts (ie. the DOE). There are many fine-grained access-control frameworks being discussed in the grid community[14] [9], but the lack of pervasive deployment of such technologies leaves the portal developer's saddled with this difficult issue.

5 Other Caveats

The VisPortal is still primarily used as a research vehicle for exploring various ways that we could deploy a production service using this paradigm. Some aspects of the portal will be used to implement services for specific NERSC users, but there are still significant impediments to this method leading to a production service. Chief among them are;

- *Difficult GUI Implementation:* JSP offers considerable advantages over CGI in terms of separating the presentation of the GUI from the web-logic, but HTML offers a very difficult and unnatural medium for designing new user interfaces. A programmer must navigate a minefield of browser quirks and incompatibilities. In addition, the HTML presentation method is entirely contrary to traditional GUI design patterns. Interfaces that try too hard to mimic their standalone/desktop counterparts are generally frustrating to use. The high latency of the stateless request/response HTML paradigm requires consideration of an entirely new approach to GUI presentation, like AMR WebSheets.

- *Debugging:* Untrapped errors in the JSP code can easily propagate deeply into the Tomcat/JSP engine or CoG kit before generating a fatal exception. Even with a good debugger, errors can be very arcane and difficult to track, thereby slowing progress on assembling new applications that have complex JSP implementations.

- *Support:* While the portal offers us a way to leverage on industry-tested web-automation platforms like JSP and Apache/SSL, and well-supported low-level middleware like Java CoG, the JSP framework that connects CoG to the JSP engine has no support mechanism. Anyone who wants to build a portal, must then take on the task of supporting the infrastructure used to implement the portal as well.

- *Middleware in Flux:* Each time the Grid middleware gets updated, numerous incompatibilities with the existing code base emerge. This limits progress as we must effectively "port" the portal to successive revision of the infrastructure. We hope that the middleware layer will stabilize in the near future.

- *Resource Management:* The portal simplifies access to large pools of remote resources, but it does not magically solve issues of resource management. In particular, how does one mediate access to very limited/shared resources like a hardware graphics pipe on an Onyx? There is also no widely available mechanism for managing fine-grained file access control for ad-hoc collaborations. Solutions are being discussed in the Global Grid Forum, but until such technology is adopted pervasively, portal developers are left with a handful of ad-hoc solutions.

6 Future and Conclusions

Over time the most successful methods explored on the VisPortal prototype will be hardened and graduated to a production portal located at the NERSC center. Eventually the portal framework will migrate to a portlet/OGSA model offered by GridSphere [12]. Whereas the current portal stores its state by serializing Java Beans to disk, GridSphere builds on more sophisticated mechanisms pioneered by the ASC portal that employ a SQL database for state management. The Open Grid Services Architecture (OGSA) offers a cleaner way to separate the form of the GUI presentation on the front-end from the function of the service implementation on the back-end of the portal. Finally, the portlet paradigm will make it easier to share GUI elements, such as a DHTML file browser, in multiple portal application contexts. We will be following the progress of GridSphere closely, but we still expect to get plenty of useful work out of the current implementation.

In the coming year, the development effort will focus on using components of the current portal to implement workflows requested by specific groups of NERSC user and their collaborators. The incorporation of workflow management is essential for the robustness of the system. Users will only adopt the portal paradigm if it simplifies their interaction with the computing resources.

Overall, we feel that web-embedded visualization really has significant potential for delivering custom remote visualization applications to supercomputing center users. It really does simplify grid software deployment when the bulk of the complex grid automation is handled by a stable centrally-controlled server. Likewise, the central point-of-presence offered by portal technology opens up many possibilities for an increased role for such technology as a nexus for integrating data management services, like replica catalogs, and data sharing mechanisms that support collaborative applications. There are still many rough edges to smooth in the technology, but we feel that current development activities in the grid portal frameworks community are targeting many of the fundamental deficiencies in the current architecture. We look forward to improving the robustness of the current architecture and using the services we have assembled so far as components that can be used for more complex workflows and applications that serve scientific collaboration in the NERSC user community.

References

[1] http://hpcf.nersc.gov/storage/hpss/probe/gfy/gfy.pdf.

[2] http://sca.ncsa.uiuc.edu.

[3] http://www.ascportal.org.

[4] http://www.nersc.gov/projects/gcm_data.

[5] http://www.rsinc.com.

[6] T. J. Jankun-Kelly, O. Kreylos, J. M. Shalf, K.-L. Ma, B. Hamann, K. I. Joy, and E. W. Bethel. Deploying web-based visual exploration tools on the grid. *IEEE Computer Graphics and Applications*, pages 40–50, 3 2003.

[7] T. J. Jankun-Kelly, K.-L. Ma, and M. Gertz. A model for the visualization exploration process. In R. J. Moorhead, M. Gross, and K. I. Joy, editors, *Proceedings of the IEEE Conference on Visualization 2002*, pages 323–330, Los Alamitos, CA, 2002. IEEE Computer Science Press.

[8] O. Kreylos, G. Weber, W. Bethel, J. Shalf, B. Hamann, and K. Joy. Remote interactive direct volume rendering of amr data (lbnl-49954). Technical report, LBNL, 2002.

[9] M. Lorch and D. Kafura. Symphony - a java-based composition and manipulation framework for computational grids. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2002.

[10] M. L. Norman, J. Shalf, S. Levy, and G. Daues. Diving deep: Data management and visualization strategies for adaptive mesh refinement simulations. *j-COMPUT-SCI-ENG*, 1(4):36–47, July/Aug. 1999.

[11] J. Novotny. The grid portal development kit. *ConcurrencyPractice and Experience*, 00:1–7, 2000.

[12] J. Novotny, M. Russell, and O. Wehrens. Gridsphere: A portal framework for building collaborations. In *1st International Workshop on Middleware for Grid Computing, Rio de Janeiro*, June 15 2003.

[13] J. Shalf and E. W. Bethel. Cactus and visapult – an ultra-high performance grid-distributed visualization architecture using connectionless protocols. *IEEE Computer Graphics and Applications*, Mar/April 2003.

[14] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, and K. Jackson. Certificate-based access control for widely distributed resources. In *Proceedings of the Eighth Usenix Security Symposium*, August 1999.

[15] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A java commodity grid kit. *Concurrency and Computation – Practice and Experience*, 13:643–662, 2001.

[16] G. von Laszewski, M. Russell, I. Foster, J. Shalf, G. Allen, G. Daues, J. Novotny, and E. Seidel. Community software development with the astrophysics simulation collaboratory. *Concurrency and Computation: Practice and Experience*, 14, december 2002.