Available online at www.sciencedirect.com

Future Generation Computer Systems 21 (2005) 3–10

FGCS
Future Generation Computer Systems

www.elsevier.com/locate/future

Incorporation of middleware and grid technologies to enhance usability in Computational Chemistry applications

Jerry P. Greenberg[a,b,1], Steve Mock[a,b], Karan Bhatia[a,b], Mason Katz[a,b], Greg Bruno[a,b], Federico Sacerdoti[a,b], Phil Papadopoulos[a,b], Kim K. Baldridge[a,b,*]

[a] *San Diego Supercomputer Center (SDSC), University of California, San Diego (UCSD) 9500 Gilman Drive, Mail Code 0505, La Jolla, CA 92093-0505, USA*
[b] *University of Zürich (UniZH), 190 Winterthurerstrasse CH-8076, Switzerland*

Available online 28 October 2004

Abstract

High performance computing, storage, visualization, and database infrastructures are increasing in complexity as scientists move towards grid-based computing. This evolution of computing infrastructure has the effect of pushing breakthrough computational capabilities beyond the reach of domain scientists. In this work, we discuss a workflow management system that allows portal construction that is fully integrated with emerging grid standards but can be dynamically reconfigured. By defining an XML schema to describe both resources, application codes and interfaces, we will enable a "pluggable" event-driven model where grid-enabled services can be composed to form elaborate pipelines of simulation, and visual analysis.
© 2004 Elsevier B.V. All rights reserved.

1. Introduction

High Performance Computing (HPC) has dramatically changed scientific research, and allowed advancements to be made at a rate far beyond that conceived a decade ago. HPC brings together the wealth of technology advancements not solely in the area of (a) hardware and raw computational speed, but also in the framework of (b) database technology, (c) visualization/environment infrastructure, (d) computational algorithms, both efficient for the latest hardware as well as integrated for enhanced capability, (e) high speed networking, and (f) remote access through new portal web service technologies. The infrastructure complexity to make these logical components work efficiently together often overwhelms domain scientists. The ultimate goal is to extend and expand efforts in interface simplification in order to allow domain scientists to build custom computational infrastructure that enables complex science to be performed.

* Corresponding author.
E-mail addresses: jpg@sdsc.edu (J.P. Greenberg), mock@sdsc.edu (S. Mock), karan@sdsc.edu (K. Bhatia), mjk@sdsc.edu (M. Katz), bruno@sdsc.edu (G. Bruno), fds@sdsc.edu (F. Sacerdoti), phil@sdsc.edu (P. Papadopoulos), kimb@sdsc.edu, kimb@oci.unizh.ch (K.K. Baldridge).
URL: http://www.sdsc.edu/ jpg (J.P. Greenberg).
[1] Tel.: +1 858 534 5138; fax: +1 858 822 5407.

0167-739X/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2004.09.027

Today, domain scientists have a myriad of technology choices available to them for building their infrastructure. In terms of computational hardware, users may choose from a broad spectrum of possibilities from loosely coupled collections of desktop PC's spread across an unknown local or remote area, to commodity cluster infrastructures, to high-end highly parallel architectures. Typically, end-users have access to a variety of different architectures, from desktop computers, to small-scale departmental Linux clusters, to high-end facilities such as those found at the NSF Supercomputer centers. While scientific users running experiments do not want to differentiate among these types of resources, its important for the scientific application developer to understand each architecture and how to extract the best performance. In this paper, we discuss some of the different architectures that we have used and how those architectures have shaped the applications.

Over the past decade, there are a number of groups developing software middleware that provides a consistent application program interface (API) for interacting with different types of resources. This *grid middleware* abstracts away some differences of the hardware but to date is still quite complex and requires significant effort in deploying and using. Using these middleware systems, new architectures, systems, and applications can be easily added to the resource list forming a "grid" of resources. One challenge, however, is to develop and execute sequences of applications with appropriate data movement done seamlessly behind the scene. Such a *composition* of applications is called "workflows". We describe our use of grid middleware for supporting Computational Chemistry and our workflow system built on top of grid middleware to support flexible user-composable application execution on grid resources.

Although advances in the capabilities of high-performance computers have made it possible for computational scientists and engineers to tackle increasingly challenging problems, at the same time, it has become considerably more difficult to build, manage, and integrate the software that can achieve the highest performance on different systems, use the resulting data most efficiently, and make it accessible to the community at large. In most cases, the rate-limiting step in pushing forward advanced chemical and biomedical research is the user interface and the protocols underlying the interface that are required to connect to the services required such as computational simulation, database access, visualization, and data dissemination. We discuss our experiences with different user environments and where we believe the technology is headed.

The successes of highly efficient, composite software, for molecular structure and dynamics prediction has driven the proliferation of computational tools and the development of first-generation computational chemistry grid-enabled infrastructure. The approach that we are taking, as illustrated in this work, is that of a layered architecture, consisting of an Interface Layer, a Middleware Layer, and a Resource Layer. The layers shield the developer from the complexity of the underlying system and provide convenient mechanisms of abstraction of functionality, interface, hardware, and software. A layered approach also helps with the logical design of a complex system by grouping together the related components while separating them into manageable parts with interfaces to join the layers together. The separation of the Resource Layer from the Middleware Layer provides the ability to design a workflow system that is not bound to a specific application or HPC resource.

2. Science methodology

GAMESS[1] is an ab initio electronic structure program, which essentially solves the molecular Schrödinger Equation to evaluate structure and properties of chemical systems. The software is distributed free of charge and is widely used internationally. The most common way data is processed and analyzed from such chemical computational programs is via the "cut and paste" method, which extracts data "by hand" from output files for purposes of visualization and other post computation analysis. Such a process is typically tedious, and if an entire class of compounds is being analyzed, the rather large effort may compromise the project. An alternative is to process data from sequential programs with command language scripts, but here as well as there are several obstacles. For example, a change in the output format of the simulation program may result in a failure of the script to process the output.

As an alternative to these approaches, we have chosen to put structured data output facilities directly into the computational program (e.g., GAMESS) as well as into the output analysis programs (e.g., PLTORB, which is used for orbital analysis). The result is that the dependency on data parsing is removed and more importantly, the data is put into a form that may be used for database storage, querying, subsequent computational process, and/or efficient transport over the grid in the form of JAVA objects for other purposes. We began our efforts towards these directions by designing an XML schema. Initially, the schema provided a template for storing only basic data, such as atomic coordinates, atom types, energies, molecular orbital coefficients, and basic input options. We have now also added additional molecular data types, for example, gradient data, Hessian data, and volumetric property grid data.

The current XML schema may be viewed at http://www.sdsc.edu/∼jpg/nmi/gamess.xsd. The production of XML data documents based on this schema was accomplished by putting directly into the GAMESS source calls to a library of C functions. We do this by using the "Castor" SourceGenerator [2] to create JAVA source that maps complex XML elements to JAVA classes. Linking GAMESS to the XML/C library together with the Java Native Interface (JNI) [3] library allows one to call JAVA methods from GAMESS.

Now, we not only have a way to store our data in a rational fashion, but also a method for transferring data to (and from) other programs. We use a very simple example—that of processing molecular orbitals. The GAMESS auxiliary program PLTORB3D uses the specifications of the calculated wave-function to create a 3D orthogonal grid of molecular orbital values. In the original implementation, two input files were required, a file containing the atomic basis set and input options, as well as molecular orbital vectors, the latter of which had to be "cut out" of the GAMESS data output file. Then, PLTORB3D produced a file that the chemistry visualization program, QMView [4,5] could read in order to display contours and isosurfaces of the molecular orbitals. With modifications to PLTORB3D similar to that made in GAMESS, PLTORB3D has now been automated eliminating much of the hand preparation and file manipulation. The modified PLTORB3D tool "unmarshalls" the XML file now written by GAMESS, adds to the JAVA object the grid volume data and "marshalls" a new XML document containing all the original data plus the volume data. This file (or JAVA object) may be viewed, stored or retrieved in order to set up a new GAMESS run, or as part of a collection queried from a database.

We have, thus, outlined a simple workflow: run GAMESS, put the results into PLTORB3D, view the orbitals with QMView, and store the data in a database. Additionally, with this basic infrastructure in place, we may easily design new workflows. For example, we could retrieve a GAMESS XML file from a geometry optimization calculation and do a "HESSIAN", i.e. vibrational mode analysis, at that geometry. Or, one could do something slightly more complex—retrieve a transition state geometry GAMESS XML file, create a "HESSIAN" analysis input on the resulting geometry, obtain the data from this run (one negative vibrational mode direction), create an input to follow the reaction path along this gradient, collecting geometries along this path for display in QMView. We have also created a non-interactive version of QMView, "QMVIEWSERVER", which can be used, for example, to read XML files, retrieve geometric coordinates, molecular orbital coefficients, property volumetric data, etc., and create GAMESS input for successive computations.

2.1. *General Middleware methodology*

The work described above is oriented towards computational chemistry and specifically to GAMESS chemistry related calculations. However, we can now easily develop a general system for facilitating scientific workflows over the grid. For example, in the above discussion of possible GAMESS related workflows, we left out any details of submitting individual GAMESS jobs, PLTORB3D jobs, QMVIEWSERVER jobs, etc., to particular platforms, or how they may be submitted in succession. What is lacking is software that integrates the individual programs, assigns and submits jobs to particular platforms, monitors the whole process, and store results.

Our previous efforts to facilitate the submission of jobs to remote platforms included the SDSC portals based on the NPACI Gridport [6] software, which provides low-level tools to GLOBUS [7] for secure access to remote platforms and to the Storage Resource Broker (SRB) [8] for storing data collections. Through the web sites built with Gridport, users were shielded from the complexity of the remote platforms used for computations and archival storage. However, building portals is non-trivial and involves the intricacies associated with writing HTML files and CGI scripts, and once built are not easily reconfigurable.

To address these problems, we are creating a general Scientific Workflow as part of the National Middleware Initiative [9]. Presently under development, the Workflow project will facilitate the building of scientific workflows from smaller tasks using web and grid services. The infrastructure is being designed to shield both users and application developers from the intricacies of the grid and specific platforms. Additionally, rather then provide a user interface such as a web browser with the Gridport portals, which limits overall flexibility for the scientist, the new infrastructure will provide "hooks" to connect user interfaces, thus, exposing an individual to a variety of user interfaces from which they can interoperate.

In order to facilitate the use of the workflow environment, a java based GUI is being developed that will allow users to connect separate applications to define a workflow (e.g., such as the GAMESS–PLTORB3D–QMView example discussed above). In more complex computations, iterative workflows can be created to "loop" until some criteria or cost-function value is reached.

Complex workflows are defined in XML documents within which the applications in the workflow are divided so as to separate high-level descriptions of the applications from the low-level execution environment details necessary to execute an application on a specific computational resource. This structure provides a layer that separates users and application developers from the underlying services. For example, a user may choose an application, specify input and not necessarily be aware of where the actual jobs are instantiated, something that is taken care of by the execution service. Data for job execution is maintained in separate XML files that specify execution paths, account information, execution protocol, and network identification. The whole process is managed by a "Workflow Engine" whereby the XML documents defining a job are instantiated as JAVA objects, resources are found, jobs are executed, and job status sent back to the user (Fig. 1).

A fundamental feature of the service briefly described here is that it is not dependent on any particular application, data file format, or operating system. A user may substitute their own user interface as long as they adhere to the protocols of the Workflow Engine. For example, a user may define their own GUI to describe a workflow, or use an available GUI to create jobs to submit to the Workflow Engine.

3. Application grids

Domain scientists have a large number of choices for building their computational infrastructure, from tightly coupled parallel architectures at one extreme to loosely coupled desktop grid systems on the other. Whichever architecture is chosen, the application code must be tailored in order to extract the best performance. GAMESS has been designed to be both portable and to exploit the capabilities of particular computational and network hardware [10]. GAMESS may be compiled with several different types of parallel libraries (MPI/LAPI, SHMEM) or may use socket con-

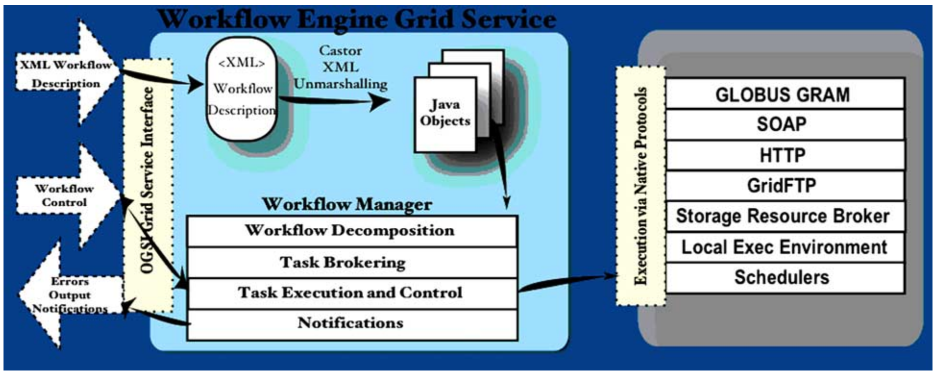


Fig. 1. The Informnet Workflow System.

nections directly. The type of communication library is hidden to the scientific programmer. For an SP cluster such as SDSC's "datastar", for example, (consisting of 8 processor P655 nodes and 32 processor P690 nodes), GAMESS is compiled with MPI. For a Linux cluster such as UniZH's "Nadelhorn" with dual processor 2.8 GHz Xeon nodes, GAMESS is compiled to use sockets where it scales reasonably well with conventional network interconnects (e.g., Ethernet). Additional modifications enable one to actually run GAMESS in more unconventional computing environments, such as a desktop grid environment. For this latter type of environment, we are investigating the use of the Berkeley Open Infrastructure for Network Computing (BOINC) [11] system for distributing computational tasks to desktop systems.

3.1. Cluster infrastructure and maintenance

Besides the grid middleware and applications described above, robust software for maintaining large clusters is essential for running computationally intense simulations. In recent years, "commodity" clusters running Linux have become quite popular because they offer individual research groups significant computing power that is relatively cheap and may be theoretically administered by the researchers themselves. In terms of chemistry applications, such as GAMESS, such clusters allow studies of complex molecular systems that involve large numbers of atoms, large basis sets, and/or higher order methods.

In practice, however, cluster administration can also be complex and time-consuming to a degree that virtually offsets the cost of the hardware. There are *n*-fold copies of the operating system to update and maintain, as well as *n*-fold systems to install new software on. If the software and the operating system are not maintained, the system may become unstable, endangering the completion of long calculations, security holes may not be patched, and software may not be updated.

The key to rapidly deploying this infrastructure is to automate the process of building and managing a cluster, which is itself a single grid end point. In our case, the automation of building the grid-enabled clustered endpoint is achieved using the NPACI ROCKS cluster distribution, which several of the authors have developed. ROCKS is a cluster-aware Linux distribution that makes it possible to deploy a world class supercomputer in a matter of hours, something that has historically taken months.

The philosophy of ROCKS is to make the installation of the operating system the basic management tool. That is, it is easier, when automated, to reinstall all nodes to a known configuration then to determine which nodes are not synchronized. This is the opposite of the maintenance procedures on desktop systems where the operating system is rarely reinstalled, or in configuration management tools such as CfEngine [12] that perform maintenance on exiting operating systems installations. Because the OS can be reinstalled in a short period of time, different application specific configurations, such as frontend, storage or compute nodes, can be easily installed and maintained and the addition of new nodes requires only a small degree of extra work.

3.2. Grid computing

The ROCKS release also contains essential software elements of grid computing. From the end-user perspective, submitting a workflow to the grid involves interaction solely with the web portal interface, a user interface to the middleware described above, or a grid scheduler such as NIMROD [13]. That is, the underlying grid components are hidden from view for day-to-day application operations. However, this underlying system involves substantial complexity and is the focus of several grid efforts today. Once a resource is selected, GLOBUS is used to start the jobs on the end point. In addition to providing the connections from a web portal to the grid end points, the grid scheduler continuously monitors all the end points for status information such as CPU utilization, free disk space, and operation system version. This information is gathered using the GLOBUS Monitoring and Discovery Service (MDS) and allows the scheduler to make reasonable scheduling decisions.

Within the last year ROCKS has matured from a cluster tool to a grid tool, and now includes the NSF Middleware Initiative's (NMI) Globus and Certificate Authority software. Because it takes only hours to build a grid-enabled cluster, the only remaining step to deploying a grid is the standard Globus certificate exchange between all end points, and the grid scheduler–portal setup.

Each individual grid end point is in itself a complex system. Fig. 2 illustrates the basic system architecture

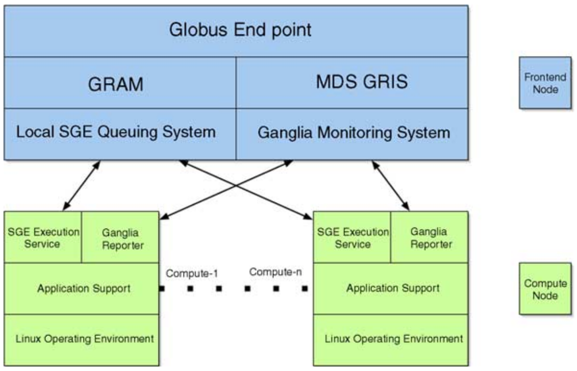
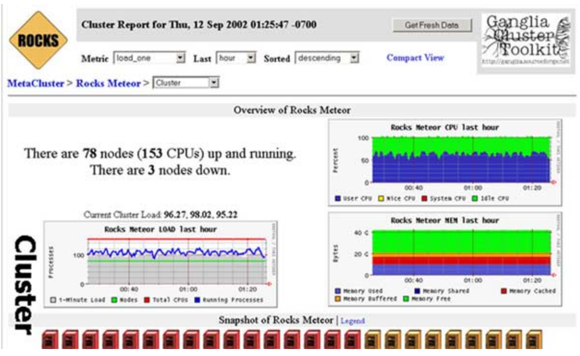


Fig. 2. Grid endpoint architecture.

of these Linux clusters. The grid scheduler is responsible for submitting jobs to the local Globus Resource Application Manager (GRAM), which next submits the job to the local cluster wide Sun Grid Engine (SGE) queuing system. The queuing system then starts the job's processes on the cluster's compute nodes, which often require supporting software and libraries to exist on the compute nodes disk. While it is possible to stage all application supporting software onto the compute nodes along with the job submission pre-staging this data often simplifies the process of grid job runs.

The final piece of software on these clustered grid end points (and included in ROCKS) is the Ganglia monitoring system [14], which contains a current snapshot of the "state" of the cluster, where state is defined by metrics such as CPU load, Memory availability, cluster size, CPU speed, etc. This Ganglia information (Fig. 3) can then be used to feed the local Grid Resource Information Service (GRIS) information about



Cluster Report for Thu, 12 Sep 2002 01:25:47 -0700

Metric load_one Last hour Sorted descending

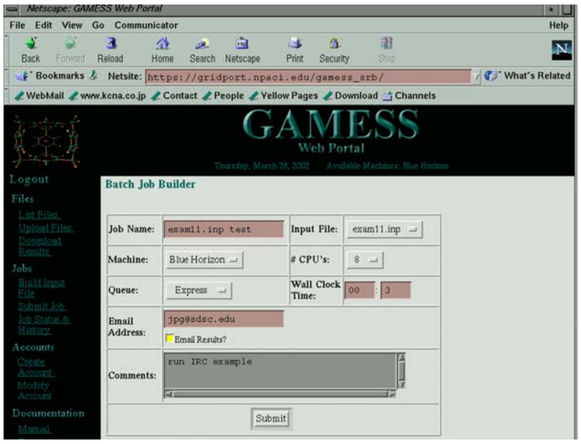
Overview of Rocks Meteor

There are 78 nodes (153 CPUs) up and running.
There are 3 nodes down.

Current Cluster Load: 96.27, 98.02, 95.22

Snapshot of Rocks Meteor | Legend

Fig. 3. The Ganglia user interface.


Netscape: GAMESS Web Portal
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Netsite: What's Related
WebMail www.kcna.co.jp Contact People Yellow Pages Download Channels
GAMESS Web Portal
Logout
Files
List Files
Upload Files
Download Results
Jobs
Build Input File
Submit Job
Job Status & History
Accounts
Create Account
Modify Account
Documentation
Manual
Submit

https://gridport.npaci.edu/gamess_srb/

Thursday, March 28, 2002 Available Machines: Blue Horizon

Batch Job Builder

Job Name:	exam11.inp test	Input File:	exam11.inp
Machine:	Blue Horizon	# CPU's:	8
Queue:	Express	Wall Clock Time:	00 : 3
Email Address:	jpg@sdsc.edu Email Results?		
Comments:	run IRC example		

Fig. 4. The GAMESS Grid Portal.

the state and configuration of the cluster. This information is then given to the grid scheduler to aid in job scheduling decisions.

Given the complexity of the computational resources and the middleware deployed, usability for end-users is a key requirement that is often lacking. Typical end-user environments require users to log in using terminal shells and execute intricate commands to the middleware system. Different resources may have different middleware systems (or different versions) and users must be proficient in order to exploit the resource.

At an early and very basic level of our work, we exploited web-based user portals to provide end-users with a convenient and familiar browser-based work environment. Using a web browser, a user logs in with a username and password and can load data, launch jobs, monitor previous jobs, and visualize the results. The complexities of all of these procedures are embedded in the portal capabilities. Toolkits such as GridPort, GridSphere [15] and OGCE [16] are developed specifically for the creation of Grid Portals. Examples include the GAMESS Portal [17] (Fig. 4).

Grid Portals, however, do not provide flexibility needed for exploratory science. They provide a fixed set of capabilities. New capabilities or workflows require significant developer effort to be done to the portal infrastructure. In addition, web-based interactions are necessarily limited in their interactivity due to the inherent limitations of HTML as a presentation language.

The next generation of end-user environments we are moving towards are so-called “thick clients”. By this we mean applications running off the end-users desktop or laptop that provide access to grid middleware through a Web Services or Grid Services API. The capabilities of the infrastructure will be dynamically revealed through discovery services, and users will be able to directly compose available applications together in innovative ways. The GAMESS XML data types and the Informnet workflow services described above are key components of such thick client architecture, are currently being developed by this team.

4. Conclusions

The successes of highly efficient, composite software, for molecular structure and dynamics prediction has driven the proliferation of computational tools and the development of first-generation computational chemistry grid-enabled infrastructure. The approach that we are taking, as illustrated in this work, is that of a layered architecture, consisting of an Interface Layer, a Middleware Layer, and a Resource Layer. The layers shield the developer from the complexity of the underlying system and provide convenient mechanisms of abstraction of functionality, interface, hardware, and software. A layered approach also helps with the logical design of a complex system by grouping together the related components while separating them into manageable parts with interfaces to join the layers together. The separation of the Resource Layer from the Middleware Layer provides the ability to design a workflow system that is not bound to a specific application or HPC resource.

With such tools, researchers can begin to ask more complex questions in a variety of contexts over a broader range of scales, using seamless transparent computing access. As more and more realistic computations are enabled at a faster turnaround time, and as problems that simply could not fit within the physical constraints of earlier generations of supercomputers become feasible, the ability to integrate methodologies becomes more critical. Assembly and modeling can often utilize different computational approaches, and are best optimized for use on different computer architectures. Thus, considerations of porting and optimizing the problem-specific application for both the high-end supercomputers, and a commodity cluster of computers often factor in.

The described technology will help to tie computation with investigator intuition regardless of location, to facilitate scientific investigations by exploiting novel grid capabilities and teraflop hardware speeds, enabling direct user input and feedback. It is anticipated that such infrastructure will impact scientists that potentially need such tools for interdisciplinary research. This will in turn foster development of new modeling, data, and computational science technologies.

Acknowledgements

We acknowledge support from the NSF through DBI-0078296 and ANI-0223043 and from the NIH through NBCR-RR08605. We acknowledge computational and administrative support from the San Diego Supercomputer Center, and the University of Zürich.

References

[1] M. Schmidt, K.K. Baldridge, J.A. Boatz, S. Elbert, M. Gordon, J.H. Jenson, S. Koeski, N. Matsunaga, K.A. Nguyen, S.J. Su, T.L. Windus, M. Dupuis, J.A. Montgomery, The general atomic and molecular electronic structure system, J. Comput. Chem. 14 (1993) 1347–1363.

[2] The Exolab Group, Castor, http://www.castor.exolab.org, 2002.

[3] S. Liang, The JavaTM native interface: programmer’s guide and specification, in: The JAVA Series, Reading, Addison Wesley Longman Inc., Massachusetts, 1999, 303 pp.

[4] K.K. Baldridge, J.P. Greenberg, QMView: a computational 3D visualization tool at the interface between molecules and man, J. Mol. Graph. 13 (1995) 63–66.

[5] K.K. Baldridge, J.P. Greenberg, QMView as a supramolecular visualization tool, in: J. Siegel (Ed.), Supramolecular Chemistry, Kluwer Academic Publishers, 1995, pp. 169–177.

[6] M. Thomas, S. Mock, M. Dahan, K. Mueller, D. Sutton, J.R. Boisseau, The gridport toolkit: a system for building grid portals, in: Proceedings of the 10th IEEE International Symposium on High Performance Computing, 2001, p. 309.

[7] I. Foster, C. Kesselman, Globus, A metacomputing infrastructure toolkit, Int. J. Supercomput. Appl. 11 (1997) 115–128.

[8] A.K. Rajasekar, M. Wan, SRB and SRBRack—components if a virtual data grid architecture, in: Proceedings of the Advanced Simulation Technologies Conference, San Diego CA, 2002.

[9] P. Papadopoulos, K. Baldridge, J. Greenberg, Integrating computational science and grid workflow management systems to create a general scientific web service environment, NSF award, 2002.

[10] R.M. Olson, M.W. Schmidt, M.S. Gordon, Enabling the efficient use of SMP clusters: the GAMESS/DDI Model, in: Supercomputing 2003, IEEE Computer Society Press, Phoenix, AZ, 2003.
[11] D.P. Anderson, BOINC: a system for public-resource computing and storage, in: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, 2004.
[12] M. Burgess, Cfengine: a site configurable engine, USENIX Computing Systems 8 (1995).
[13] D. Abramson, A. Lewis, T. Peachy, Nimrod/O: a tool for automatic design optimization, in: Proceedings of the 4th International Conference on Algorithms & Architectures for Parallel Processing (ICA3PP 2000), Hong Kong, 2000.
[14] F.D. Sacerdoti, M.J. Katz, M.L. Massie, D.E. Culler, Wide area cluster monitoring with ganglia, in: Proceedings of the IEEE Cluster 2003 Conference, Hong Kong, China, 2003.
[15] J. Novotny, M. Russell, O. Wehrens, GridSphere: an advanced portal framework, in: Euromicro 2004, Rennes, France, 2004.
[16] OGCE: Open Grid Computing Environment, http://www.ogce.org, 2004.
[17] GAMESS Portal, http://www.gridport.npaci.edu/gamess/, 2004.

Jerry Greenberg graduated with a Bachelors Degree in Chemistry from UCLA in 1978. He did his graduate work at UCSD, earning a Masters Degree in 1981 and a PhD in chemistry in 1986. He has been a Staff Scientist at the San Diego Supercomputer Center from 1988 to the present.