The XCAT Science Portal *

Sriram Krishnan, Randall Bramley, Dennis Gannon, Madhusudhan Govindaraju,
Rahul Indurkar, Aleksander Slominski, Benjamin Temko
Department of Computer Science, Indiana University, Bloomington, IN

Richard Alkire, Timothy Drews, Eric Webb
Department of Chemical Engineering, University of Illinois, Urbana-Champaign, IL

Jay Alameda
National Computational Science Alliance, IL

Abstract

The design and prototype implementation of the XCAT Grid Science Portal is described in this paper. The portal lets grid application programmers easily script complex distributed computations and package these applications with simple interfaces for others to use. Each application is packaged as a "notebook" which consists of web pages and editable parameterized scripts. The portal is a workstation-based specialized web server, capable of executing the application scripts and launching remote grid applications for the user. The portal server can receive event streams published by the application and grid resource information published by Network Weather Service (NWS) [32] or Autopilot [15] sensors. Notebooks can be "published" and stored in web based archives for others to retrieve and modify. The XCAT Grid Science Portal has been tested with various applications, including the distributed simulation of chemical processes in semiconductor manufacturing and collaboratory support for X-ray crystallographers.

Keywords : Grid, Science Portal, Distributed Simulations, Scripted Applications.

1 Introduction

The concept of a Science Portal was first introduced by the National Computational Science Alliance (NCSA) as part of a project designed to provide computational biologists with access to advanced tools and databases that could be shared by a community of users via web technology. A Science Portal can be broadly defined as an application specific environment for using and programming complex tasks involving remote resources. Over the past year the Science Portal concept has been heavily influenced by the emergence of the Grid [12] as a computational platform.

A Grid is a set of distributed services and protocols that have been deployed across a large set of resources. These services include authentication, authorization, security, namespaces and file/object management, events, resource co-scheduling, user services, network quality of service, and information/directory services. Together these services enable applications to access and manage the remote resources and computations. Web-based Grid Portals provide mechanisms to launch and manage jobs on the grid, via the web. Grid Science Portals are problem solving environments that allow scientists the ability to program, access and execute distributed applications using grid resources which are launched and managed by a conventional Web browser and other desktop tools. In such portals, scientific domain knowledge and tools are presented to the user in terms of the application science, and not in terms of complex distributed computing protocols. The system effectively makes the grid into a vast and powerful computation engine that seamlessly extends

*Supported by NSF grants 4029710 and 4029713

the user's desktop to remote resources like compute servers, data sources and on-line instruments.

This paper describes the XCAT Science Portal (XCAT-SP) which is an implementation of the NCSA Grid Science Portal concept. XCAT-SP is based on the idea of an "active document" which can be thought of as a "notebook" containing pages of text and graphics describing the science of a particular computational application and pages of parameterized, executable scripts. These scripts launch and manage the computation on the grid, and results are dynamically added to the document in the form of data or links to output results and event traces.

XCAT-SP is a tool which allows the user to read, edit, and execute these notebook documents. The goal of this research and the focus of this paper is to address the following set of questions.

- How well does the active document model work for real scientific applications?
- How does one use scripts to steer computations from the portal?
- What is a simple and efficient mechanism to store and retrieve data specific to each application?
- How should the portal be designed to interact with an event system to receive feedback from the remotely executing applications?
- How can a portal use a grid monitoring system to provide resource utilization information about its environment?

In the following sections we first survey current Grid Portal designs, and then describe the XCAT-SP architecture in detail. We conclude with a look at examples of its use.

2 Existing Grid Portals

The area of Grid Portal design is now an extremely active and important part of the emerging Grid research agenda. The existing projects can be grouped into three categories :

- User Portals for simple job submission and tracking, file management and resource selection
- Portal Construction Kits, that provide the APIs necessary for a portal to communicate with Grid services
- Science Portals

In the user portal category, the NPACI Hot Page [28] is the first and most successful system. Other user portal projects are the European project Unicore [4], Nimrod-G from Australia [11], and the IPG Launch Pad, which is the user portal for NASA's Information Power Grid [6] .

At least three projects provide portal construction toolkits. The Argonne Commodity Grid (CoG) [17] toolkit is a Java interface for Globus. GPDK from Lawrence Berkeley Labs [8] is a JSP API for CoG, and JiPANG from Tokyo Institute of Technology [25], uses Sun Microsystem's Jini [23] to provide an interface to both CoG and networked solvers like Ninf [26] and Netsolve [7].

Science Portals have a variety of forms. Some are designed around relatively specific application domains. For example, the Cactus Portal [16] from the Albert Einstein Institute was originally designed for black hole simulations and the ECCE/ELN [27] project from ORNL, LBNL and PNNL is for Computational Chemical Engineering. The Lattice Portal [20] from Jefferson Labs is a user portal for high-energy physics. One category of science portals directly addresses the problem of building multidisciplinary applications. The Gateway project [5] and the Mississippi project [31] use CORBA [14] and Enterprise Java Beans (EJB) [24] to build a three-tier architecture for launching and scheduling multiple applications. These two projects also use scripting to orchestrate large, complex application scenarios. Another CORBA-based project is the Rutgers Discover portal [10] which also provides a good interface for computational steering and collaborations.

3 The XCAT Science Portal

An initial prototype science portal that tests some of the features described above has been developed over the last year at Indiana University with help from NCSA and their Chemical Engineering Team. The portal differs radically in its architecture from the examples described above because it does not use a centralized web server on a remote machine. In our system the portal software that runs on each user's desktop/laptop has a built-in server. The reason for this is that the XCAT Science Portal is designed to integrate the user's desktop environment with the remote grid resources. If the portal resides elsewhere, the only tool the user can use to interact with the Grid is a Web browser or other HTTP clients. In our model, the portal server provides a single, local gateway between the Grid Services and local applications. A local web browser can still interact with it through HTTP, but other applications may possibly communicate with it via local protocols and services, such as COM [22], .NET [21] and Bonobo/Gnome [1].

As illustrated in Figure 1, the major components of the portal server include:

- A Java-based (Tomcat) server engine driven by a set of Java Servlets that manage access to the other components.
- A *notebook* database. A *notebook* is an active document defined by an XML object that describes a set of resources used in a computational application. It consists of documents, web pages, execution scripts, and other notebooks.
- A *Script Engine* that is used to execute complex Grid operations. The scripting is currently in JPython, which has become popular with many computational scientists. We provide JPython-based interfaces to the Argonne CoG toolkit, which in turn, provides access to Globus functionality and the GSI [18] Grid authentication mechanisms. It also has an API that allows easy access to the DOE Common Component Architecture (CCA) services [19].
- An *Event Subsystem* that is capable of handling event messages, which may be generated by grid resources or user applications.
- A *Grid Performance Monitor* that provides the user with a view of available resources, their current loads and network loads.
- A *Remote File Management Interface* that uses the GSI enabled FTP service.

3.1 The Notebook Database

The underlying directory structure of the filesystem is used as the database to support the portal. The database stores a notebook corresponding to each computational application. Each notebook is stored as a directory and each page of the notebook is stored in a different subdirectory. An XML file containing meta-data about the notebook and a list of pointers and references to the pages in the notebook is also stored in the local database. Figure 2 shows a snippet of such an XML file. It describes a notebook session, with a title *Notebook_Intro*, containing a notebook page, *BigPicture*.

A notebook session can be saved as a jar (Java archive) file, which can be published in a repository using GSI-enabled FTP or other file transfer services. Authorized portal users can retrieve the jar file from the repository and place it in a database local to their own portal. This enables the portal users in a scientific community to share data corresponding to their experiments with their peers.

3.2 Grid Application Scripting

One difference between a *user portal* and a *science portal* is the complexity of the tasks that the portal supports. A user portal allows users to submit single jobs to the grid. The portal provides features to make it very simple to manage the job, providing load-time and run-time information, and to help the user select resources and to monitor the execution of the job. In a science portal, the applications tend to be more complex. A single scientific experiment may involve running many different computational simulations and data analysis tasks. It may involve coupled multidisciplinary applications, collaboration, and remote software components linked together to form a distributed application. Often these complex tasks may take

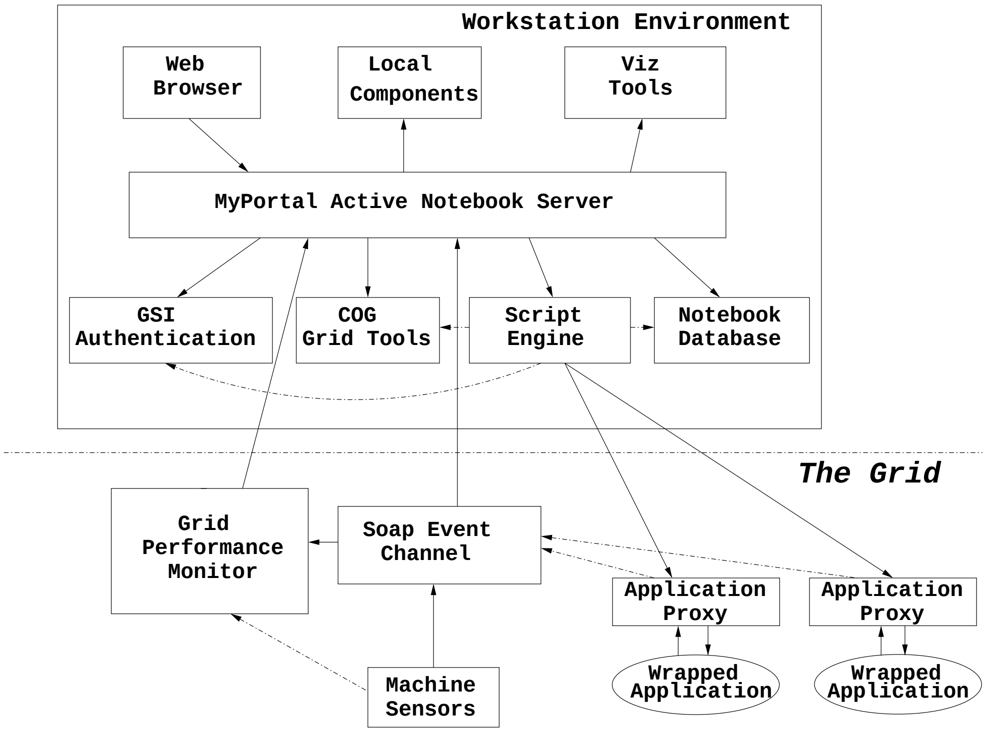

Figure 1: The Big Picture

a great deal of effort to plan and orchestrate, and the entire application may need to be run many times each with a slightly different set of parameter values. We have found that the best way to allow this sort of computation to be carried out is to allow the scientist access to a simple scripting language which has been endowed with a library of utilities to manage Grid applications. Furthermore, we provide a simple tool which allows the scientist to build a web form interface to configure and launch the scripts. Users of the scripts simply fill in parameter values to the web form and then hit the *Submit* button. This launches a script which executes on the user's desktop, but manages remote applications on the grid. Our prototype implementation uses the (J)Python language for scripting because it is popular with scientists and has an excellent interface to Java, and we make the scripts grid-enabled by providing an API to Globus Services using the Cog Toolkit.

Figure 4 illustrates a portal interface, which is typically application-dependent and is configurable by the users. In the panel on the left, there is a view of an open notebook session. It consists of a set of pages and script forms. In this figure, the form for a simple script which launches a local visualization application is shown. Parameter values selected by the user from the form page are bound to variables in the script. By selecting *Edit* both the script and the form page may be edited as shown in Figure 5. In this case, the script launches a local program called *animator* which takes as a parameter the name of a simulation output file to animate. In this example the script is trivial, but it is not much more difficult to write a script to launch an application on the grid and to manage remote files.

A second form of scripting is used to manage the local details of the program's execution on the Grid. The remote applications are managed by *application managers*. In most cases, the applications that the scientists and engineers want to run on the Grid are not *grid aware*, i.e. they are ordinary programs that read and write data to and from files. In some cases, we have access to the application source, but often that is not available - e.g, when using commercial applications codes. An application manager is an agent process that helps the application make use of grid services. For example, the manager can stage input files from remote locations or invoke post-processing on the application output when the application has finished. The manager also serves as an event conduit between the application and the portal. If the application dies or

```xml
<activeNotebook xmlns="http://www.extreme.indiana.edu/an">
        <activeNotebookInfo>
                <title>Notebook_Intro (session)</title>
                <creationDate>Thu Apr 19 10:54:10 EST 2001</creationDate>
                <modifiedDate>Thu Apr 19 10:54:18 EST 2001</modifiedDate>
                <version>1.0</version>
                <id>NotebookIntro.7444</id>
                <open>true</open>
                <relatedTo>NotebookIntro</relatedTo>
                <unsaved>false</unsaved>
        </activeNotebookInfo>
        <pageContent>
                <title>BigPicture</title>
                <url>/an/database/notebook/nNotebookIntro.7444/
                     pBigPicture/big_picture.html</url>
                <id>BigPicture</id>
                <number>1</number>
                <open>false</open>
        </pageContent>
</activeNotebook>
```

Figure 2: An XML file with notebook metadata

creates a file, the manager can send an event back to the portal indicating that this has happened. The application manager is shown in Figure 6.

The application manager can also act as a service broker for the application. The manager can register itself with the Grid Information Service[13] and advertise the application's capabilities. If a user with the appropriate authorization discovers it, then the manager can launch the application on behalf of the user and mediate the interaction with the user. For example, suppose the application is a library for solving sparse linear systems of equations on a large parallel supercomputer. The manager can export a remote solver interface that takes a sparse linear system as input and returns solution vectors and error flags as output. If a user has a remote reference to the manager, the solver can be invoked by a remote method call passing a linear system (or its URI) as a parameter and the solution vector can be received as a result of the call. This is the model used by JiPANG to invoke Ninf and Netsolve.

In the XCAT system, the application managers conform to the DOE Common Component Architecture(CCA) specification. XCAT is our implementation of the CCA specification, built on top of SoapRMI [29], that allows the users to write CCA compliant components, in C++ and Java. The application managers are designed to be *scriptable* components, which have one standard port providing the creator with the ability to download a script which the component can run. The scripting language and library used by the component is identical to the language and library available to the portal engine. The application managers combine the advantages of a persistent remote shell with that of a remote object which may be invoked through a well defined set of interfaces. Furthermore, the interfaces that a manager component supports can change dynamically by simply downloading a new script. This allows the portal to dynamically change the behavior of a remote application to suit new problems or requirements.

XCAT provides a JPython based scripting interface to instantiate remote components, wire them together using input and output ports and orchestrate the computations. The portal uses this scripting interface, whose API is described in Figure 8.

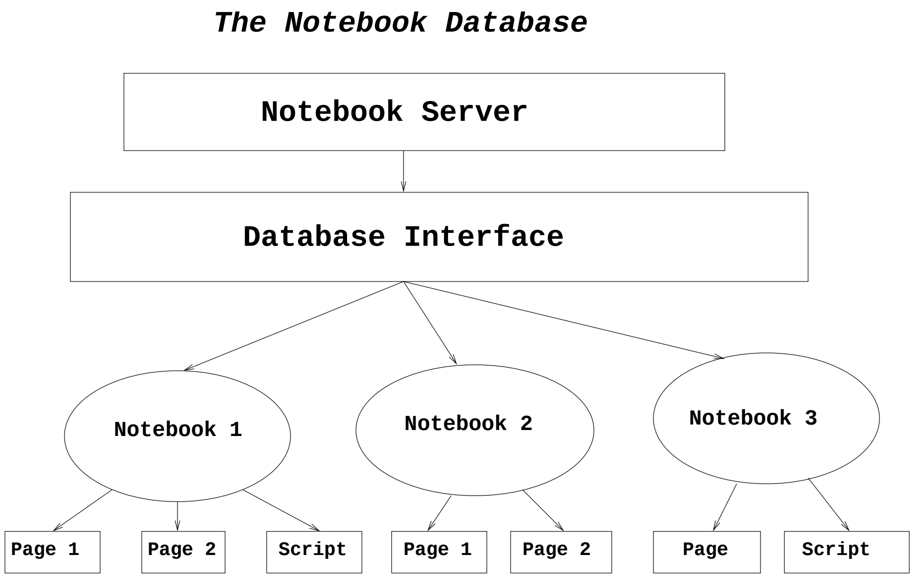

Figure 3: The Notebook Database

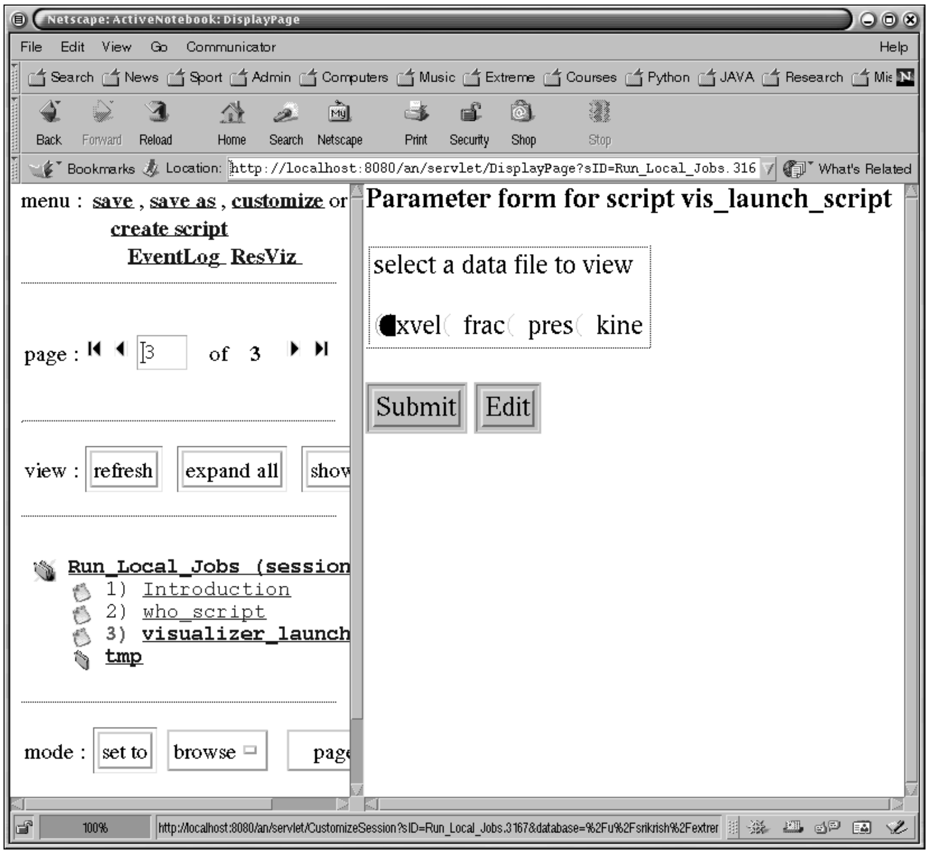

Figure 4: Snapshot of XCAT SP

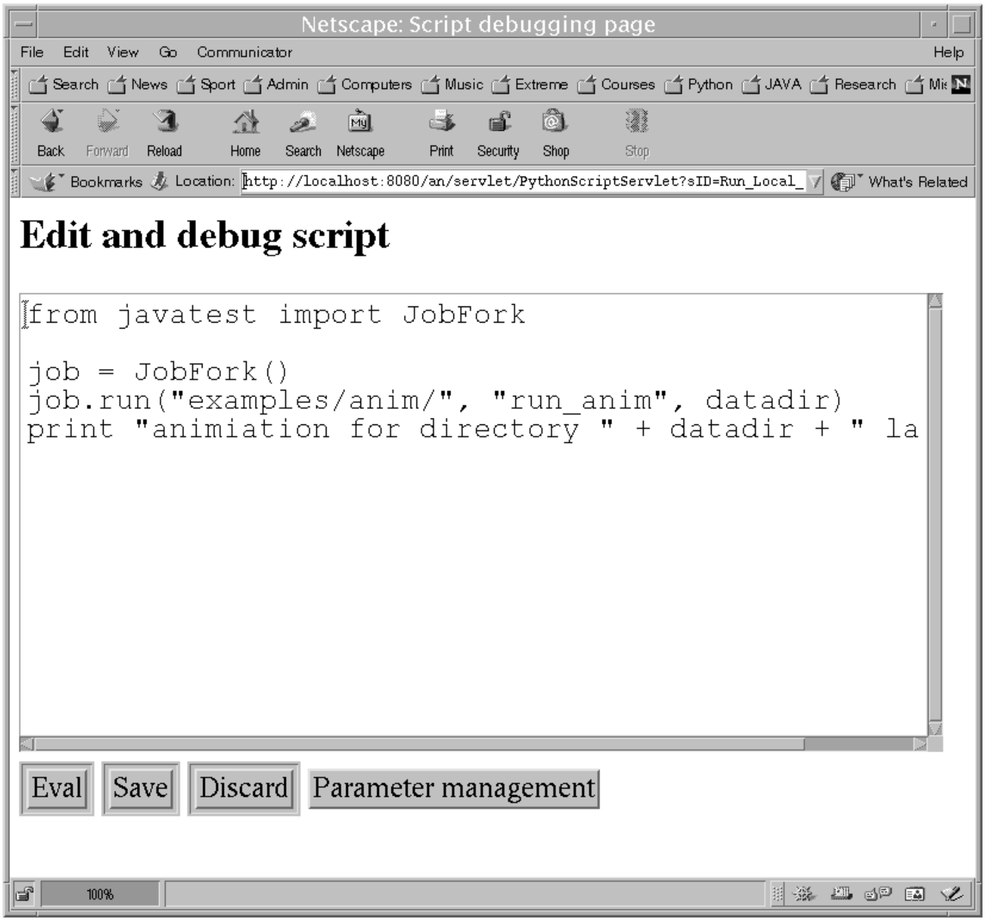

Figure 5: Script Page

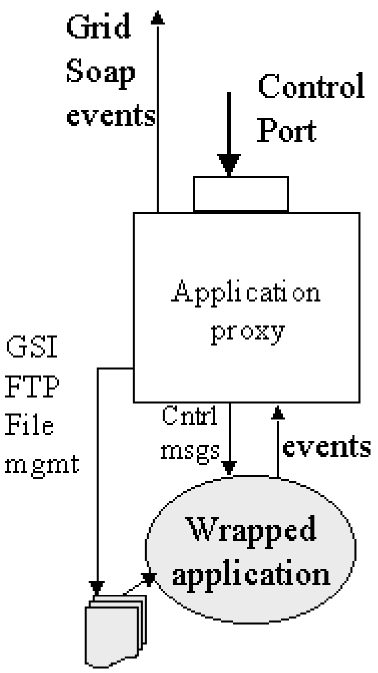

Figure 6: XCAT Application Managers

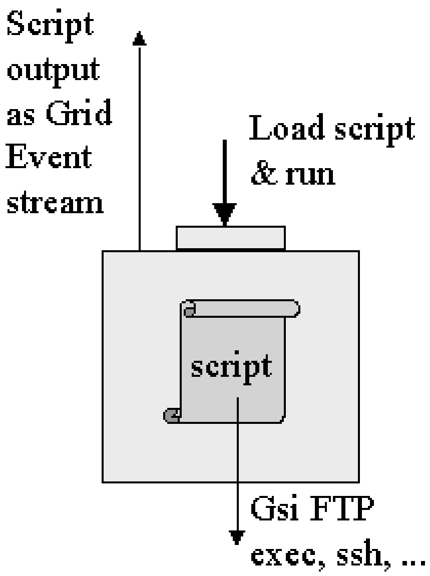

Figure 7: Scriptability of Application Managers

```
def createComponent (componentName):
def setMachineName  (componentModel, machineName):
def setCreationMechanism (componentModel, creationMechanism):
def createInstance (componentModel):
def connectPorts (outputPortComponent, outputPortName,
                  inputPortComponent, inputPortName):
def start (componentModel):
def kill (componentModel):
def invokeMethodOnComponent (componentModel, portClassName, portName,
                             methodName, methodParams):
```

Figure 8: JPython API to XCAT

3.3 Event Subsystem

The XCAT Science Portal uses the SOAP Events system [30] to decouple communication between the scripting engine and the remote jobs launched by the portal. This decoupling is important so that applications can continue when the portal itself is shut down, and to provide robustness to the overall system. A remote job can be an instantiated component that reads and writes files or a Grid Monitoring Architecture (GMA) [2] that collects data for fault detection and performance tuning from a computational grid. Such systems can indicate their progress by sending out events at intervals to interested listeners.

The SOAP Events system is based on SoapRMI, which uses HTTP as the network protocol and SOAP 1.1 [3] XML-Schemas as the wire protocol. By using SoapRMI, the portal can receive events from any SOAP 1.1 system. Events can be published by simply writing preformatted strings onto a socket, allowing frameworks that use different languages and platforms to easily publish events in an XML format. However, to receive asynchronous event notification, a system is required to continually renew its lease with the event publisher. To alleviate this, our event system provides a publisher based on a "pull-model" that allows connections from simple listeners. As a result, the portal does not need a dedicated thread to renew a lease with the event channel.

The portal is designed to leverage the advanced features of the event system: firewall friendliness, directory services and failsafe mechanisms. The portal registers itself with a well-known SOAP event channel as a listener. The processes that are launched by Python scripts can provide information about the status of their computations by publishing events to the event channel. Thus the channel multiplexes and demultiplexes events by connecting the portal with different event publishers. The event channel can provide type-based filtering, allowing the portal to register interest in events only of specific patterns.

The portal registers with the well-known SOAP event channel via the scripting engine. To register as a subscriber, the scriptwriter provides a JPython object that implements the EventSubscriber Interface specified by the SoapRMI Events API. This object is then registered with the event channel. Figure 9 shows fragments of JPython code illustrating this process.

Figure 10 shows an example of an XML resource event. The portal uses SoapRMI to receive this event as a Java object with accessor methods to retrieve the values of the fields of the event. The portal can use this information to process the event and render it on a browser if needed. The XML-Schema event system lends itself to extensibility and self-describing event formats, thus making it possible for the portal to interoperate with a wide variety of other event systems, including the NWS, Autopilot sensors, and application-level events.

```
import sys

# import EventSubscriber, from extreme.soaprmi.events package
from extreme.soaprmi.events import EventSubscriber

# import Receiver, which acts as the EventChannel
from extreme.soaprmi.events import EventChannel

# A specialization of the generic EventSubscriber
class MyEventSubscriber(EventSubscriber):

    def __init__(self):
        # constructor code goes here

    # This is the only method in the EventSubscriber Interface
    def handle(self,event):
        # however you wish to handle individual events

# code to actually register with an event channel
def subscribe():

    # get handle to the event channel
    channel = ...

    # register with the event channel
    eventSubscriber = MyEventSubscriber()
    channel.subscribe(eventSubscriber)
```

Figure 9: Subscribing to an Event Channel

3.4 The Grid Performance Monitor

The Grid Performance Monitor (GPM) uses the event subsystem to provide a user with a list and visualization of available resources and the current and predicted future loads on these resources. The data for these loads is obtained from the NWS. The GPM is designed as a thin layer on top of SoapRMI events and an event channel. This provides the portal with the flexibility to cooperate and exchange signals/events with a vast variety of event generators.

The event visualizer component of the GPM subscribes to the event channel. The visualizer registers interest only in those events that are for resource utilization or a related type. The lease based subscription mechanism of the channel provides robustness in the face of failures of components in any distributed system. The event channel even obviates the need for a global registry of all sensors. Sensors send events to the event channel at periodic intervals. Detection of a stoppage in events from a particular sensor can be determined to be due to the failure of the sensor. Event generators that send events at irregular intervals can be required to send simple heart-beat events at regular intervals to indicate that they are still operational.

3.5 Security & Authentication

Authentication is handled via the Globus GSI. The user can either use local globus credentials on the portal's server via the Globus CoG Kit, or can remotely upload credentials into the portal via the MyProxy [9] CoG Kit. The initial startup screen has text fields for the user to enter in the appropriate information: his/her

```
<MachineUtilizationEvent>
<eventNamespace>http://www.extreme.indiana.edu/soap/events/
                 resdat#MachineUtilizationEvent </eventNamespace>
  <eventType>resdata.machine.utilization</eventType>
  <source>rainier.extreme.indiana.edu</source>
  <timestamp>982862312897<timestamp>
  <cpuUtilization>0.88<cpuUtilization>
  <memoryUsed>123988</memoryUsed>
</MachineUtilizationEvent>
```

Figure 10: An example Resource-Event in XML.

globus credential password for a local credential, or a server, tag name and password for a MyProxy credential. In either case, the portal server loads a GlobusProxy object from the relevant source for use in authentication and instantiation. If Cookies are enabled by the user, the server sets a Cookie object in the user's browser that maps the session to the Proxy so that, when the user leaves the site, his/her identity isn't lost. This helps the portal do better session management. Even if the user has disabled the use of Cookies, the portal works fine, although it loses some of its session tracking capabilities.

4 Applications

4.1 NCSA Chemical Engineering Applications

The work done with the Chemical Engineering team from NCSA is an example of the kind of science problems the portal is intended to solve. The simulation models the process of copper electrodeposition in a submicron sized trench which is used to form the interconnection on microprocessor chips. The simulation consists of two linked codes. One consists of a continuum model of the convective diffusion processes in the deposition bath adjacent to the trench. The second consists of a Monte Carlo model of events that occur in the near-surface region where solution additives influence the evolution of deposit shape and roughness during filling of the trench. The codes communicate by sharing data files about common boundary conditions. Figure 11 shows the coupled codes and the filter that is added to insure stability of the linked computational system.

The codes are run separately on the Grid. The transfer of files is done using grid based file-management and transfer utilities. The interface to the Grid is provided by "Application Managers". As described before, these are wrappers which provide access to grid services such as GSI, grid-events, etc. to the codes and make them grid-aware. Each execution is set-up and controlled from the controlling JPython script which runs inside the portal. The primary mechanism for getting feedback is the event system. Grid file-management tools can be used to transfer output files which are generated. Events which come back from the applications are handed off to event handlers which have been registered, or are logged. Special events could be used to trigger off event handlers which can change or control the course of the execution.

This application illustrates several interesting scenarios in collaboration. The experiment is set up by the Chemical Engineers using the tools provided in the portal. Simple web forms are created for parameter input which will control the experiment. An example of one such form is shown in Figure 13. Subsequent users do not need to know about these parameters or the mechanics of the grid computation. They will interact with only the portal web interface and event notification mechanisms.

4.2 IU Xports project

A second application is a collaboratory for X-ray crystallographers using the beam lines at Argonne's Advanced Photon Source (APS) and Lawrence Berkeley's Advanced Light Source (ALS). This work will allow

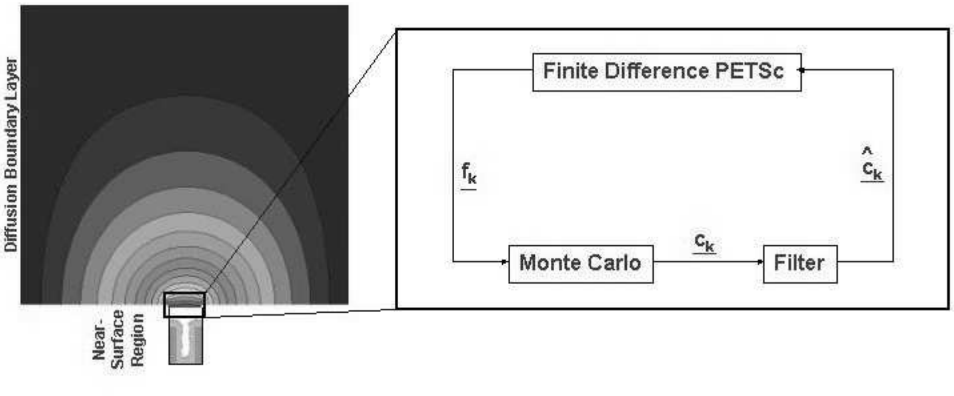

Figure 11: Linked Chemical Engineering Codes

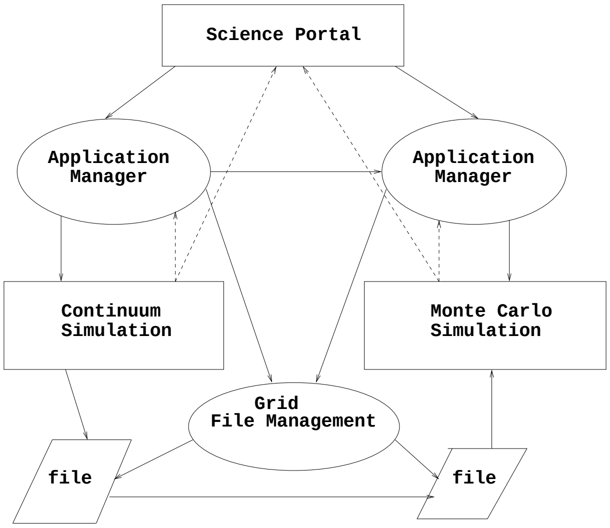

Figure 12: Chemical Engineering Application Setup

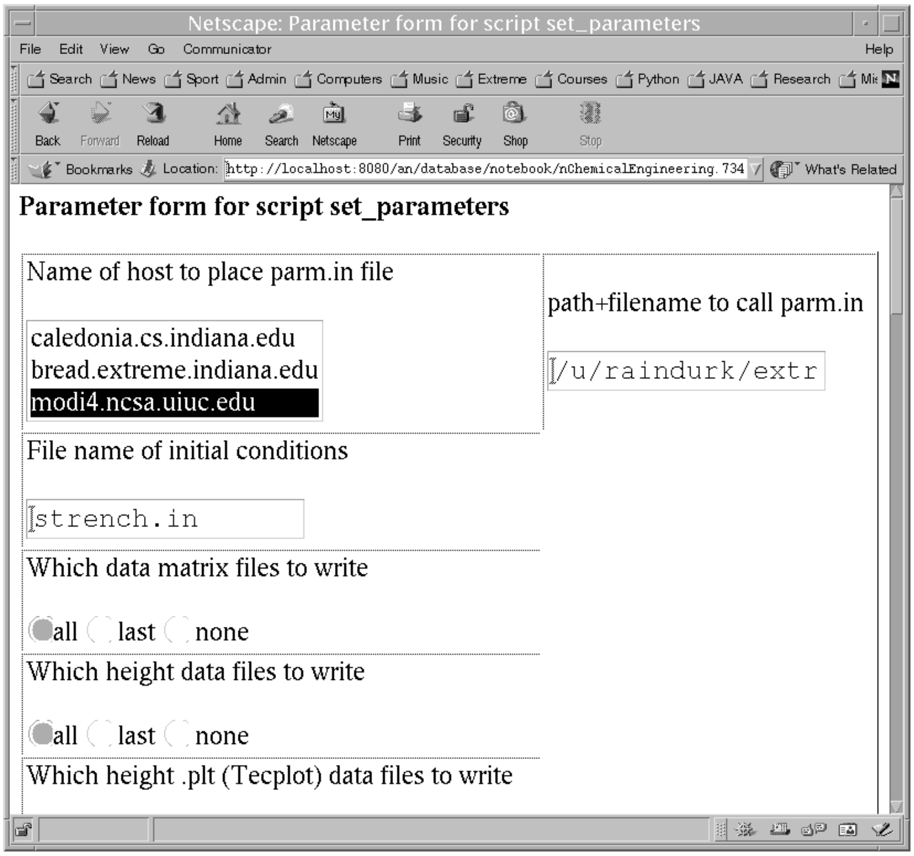

Figure 13: Parameter Form

users at remote laboratories to send sample crystals to the beam lines, collaborate with the scientists preparing and mounting the sample, then to receive initial images of the execution, over the network. They can then dynamically upload new control parameters or, if the sample appears flawed, terminate the run. In addition to large amounts of data (up to a Terabyte/day) and numbers of files (1-3 per second) this application requires multiple video streams, accessing high-speed research networks, and synchronous geographically distributed collaboration.

The portal was used to launch part of the experimental setup from the client site. Using the JPython controlling script and the Java Application Managers, local applications were launched and controlled. The setup of the experiment closely resembled that of the Chemical Engineering one. Events were used to get feedback on the progress of the execution.

5 Conclusions

This paper has described the XCAT Science Portal system. The contributions of this research project include

- providing a programming tool for grid application designers that allows them to script complex applications and package them as "active document" notebooks for others to use.
- demonstrating how a grid event system can be integrated into both the grid applications and resource monitoring to provide the user with important feedback about the runtime behavior of his or her applications.
- showing that a distributed software component architecture (in this case the DOE CCA model) can be used as an effective tool to manage distributed applications based on legacy software, which is not grid-aware.

Future work includes integration of the resource and component directory services with the Grid Forum standards for information services and with the emerging work on the Web Service Directory Language (WSDL) that is being advocated by industry groups. In addition, we are building interfaces to intelligent resource brokers and building components that are capable of adapting to available grid resources.

References

[1] GNOME, visited 4-1-2001. www.gnome.org.

[2] Brian Tierney et al. White paper: A grid monitoring service architecture (draft), visited 03-10-01. http://www-didc.lbl.gov/GridPerf/papers/GMA.pdf.

[3] D. Box et al. Simple Object Access Protocol 1.1. Technical report, W3C, 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[4] Dietmar Ervin et al. The Unicore HPC Portal, visited 04-25-2001. http://www.unicore.de/.

[5] Geoffrey Fox et al. Gateway Computational Portal, visited 04-27-01. http://www.gatewayportal.org/.

[6] George Myers et al. NASA IPG Launch Pad Portal, visited 04-27-2001. http://www.ipg.nasa.gov/.

[7] Jack Dongarra et al. Netsolve, visited 04-27-01. http://www.cs.utk.edu/netsolve/.

[8] Jason Novotny et al. The grid portal development kit, visited 04-01-01. http://dast.nlanr.net/Features/GridPortal/.

[9] Jason Novotny et al. Myproxy, visited 04-12-01. http://dast.nlanr.net/Features/MyProxy/.

[10] Manish Parashar et al. DISCOVER, visited 04-27-01. http://www.discoverportal.org/.

[11] Rajkumar Buyya et al. Nimrod, A Tool for Distributed Parametric Modelling , visited 04-25-2001. http://www.csse.monash.edu.au/ davida/nimrod.html/.

[12] Ian Foster and Carl Kesselman. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.

[13] Grid Forum Information Services Working Group. GGF GIS Working Group Charter, visited 06-29-01. http://www-unix.mcs.anl.gov/gridforum/gis/.

[14] Object Management Group. The Common Object Request Broker: Architecture and specification, July 1995. Revision 2.0.

[15] Pablo Group. AutoPilot, visited 04-01-2001. http://www-pablo.cs.uiuc.edu/Project/Autopilot/AutopilotOverview.htm.

[16] Albert Einstein Institute. Cactus, visited 04-27-01. http://www.cactuscode.org/.

[17] Argonne National Lab. CoG, visited 04-12-2001. http://www.globus.org/cog.

[18] Argonne National Lab. GSI, visited 04-12-2001. http://www-fp.globus.org/security/v1.1/.

[19] Argonne National Laboratory, Indiana Univeristy, The Advanced Computing Laboratory at Los Alamos National Laboratory, Lawrence Livermore National Lab, and Univeristy of Utah. Common Component Architectue, visited 1-10-2000. http://z.ca.sandia.gov/ cca-forum see also http://www.extreme.indiana.edu/ccat.

[20] Jefferson Labs. Lattice Portal, visited 04-27-01. http://lqcd.jlab.org/.

[21] Microsoft. .NET framework, visited 02-10-01. http://www.microsoft.com/net/.

[22] Microsoft. COM, visited 4-2-2001. http://www.microsoft.com/com.

[23] Sun Microsystems. Jini, visited 3-1-2001. http://www.sun.com/jini.

[24] Sun Microsystems. EJB, visited 7-15-99. http://java.sun.com/products/ejb/index.html.

[25] Tokyo Institute of Technology. JiPang, visited 04-27-01. http://matsu-www.is.titech.ac.jp/ suzumura/jipang/.

[26] Tokyo Institute of Technology. Ninf, visited 04-27-01. http://ninf.etl.go.jp.

[27] ORNL, LBNL, and PNNL. DOE2000 Electronic Notebook, visited 04-27-01. http://www.emsl.pnl.gov:2080/docs/collab/research/ENResearch.html.

[28] San Diego Supercomputer Center (SDSC), the University of Texas (UT), and the University of Michigan (UM). NPACI Hot Page, visited 04-25-2001. https://hotpage.npaci.edu/.

[29] A. Slominski, M. Govindaraju, D. Gannon, and R. Bramley. Design of an XML based Interoperable RMI System : SoapRMI C++/Java 1.1. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Pages 1661-1667*, June 25-28 2001.

[30] Aleksander Slominski, Madhusudhan Govindaraju, Dennis Gannon, and Randall Bramley. SoapRMI Events: Design and Implementation. Technical Report TR-549, Indiana University, April 2001.

[31] Mississippi State University. The Mississippi Computational Web Portal, visited 04-27-01. http://WWW.ERC.MsState.Edu/labs/mssl/mcwp/.

[32] Rich Wolski, Neil T. Spring, and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 1999. also UCSD Technical Report Number TR-CS98-599, September, 1998.