The Legion Grid Portal

Anand Natrajan, Anh Nguyen-Tuong, Marty A. Humphrey, Michael Herrick, Brian P. Clarke, Andrew S. Grimshaw

Abstract — **The Legion Grid Portal is an interface to a grid system. Users interact with the portal, and hence a grid through an intuitive interface from which they can view files, submit and monitor runs, and view accounting information. The architecture of the portal is designed to accommodate multiple diverse grid infrastructures, legacy systems and application-specific interfaces. The current implementation of the Legion Grid Portal is with familiar web technologies over the Legion grid infrastructure. The portal can be extended in a number of directions — additional support for grid administrators, greater number of application-specific interfaces, interoperability between grid infrastructures, and interfaces for programming support. The portal has been in operation since February 2000 on *npacinet*, a worldwide grid managed by Legion on NPACI resources.**

I. Overview

The Legion Grid Portal is a grid computing environment project designed to make grids accessible to users *via* easy-to-use interfaces. The portal uses standard, off-the-shelf software in conjunction with existing grid infrastructures to facilitate access to a grid. In its current implementation, the portal employs Legion [13] as the underlying grid infrastructure by using Legion's command-line tools. In other words, when a user interacts with the browser, the appropriate tool with the appropriate parameters is invoked at the back-end. The portal can support the full suite of Legion command-line tools; in practice, it supports a rich subset of the existing tools. Particularly, it supports initiating and monitoring runs of grid applications and accessing the Legion distributed file system. Services such as security, scheduling, data transfer, etc. are supported implicitly. The portal is in operation and is servicing the needs of users of *npacinet*, Legion's worldwide grid.

Manuscript received June 30, 2001. This work supported in part by DARPA (Navy) contract #N66001-96-C-8527, DOE contract DE-FD02-96ER25290, DOE contract Sandia LD-9391, DOE contract D459000-16-3C, DARPA contract SC H607305A, Logicon (for the DoD HPCMOD/PET program through the NAVO MSRC) contract DAHC 94-96-C-0008, National Science Foundation Next Generation Software grant EIA-9974968, National Science Foundation NPACI grant ASC-96-10920, and a grant from NASA-IPG.

Anand Natrajan is with the Dept. of Computer Science at the University of Virginia, Charlottesville, VA 22904-4740, USA (e-mail: anand@virginia.edu).

Anh Nguyen-Tuong is with the Avaki Corporation, Charlottesville, VA 22902, USA (e-mail: anh@avaki.com).

Marty A. Humphrey is with the Dept. of Computer Science at the University of Virginia, Charlottesville, VA 22904-4740, USA (e-mail: humphrey@cs.virginia.edu).

Michael Herrick is with the Avaki Corporation, Charlottesville, VA 22902, USA (e-mail: mherrick@avaki.com).

Brian P. Clarke is with the Dept. of Computer Science at the University of Virginia, Charlottesville, VA 22904-4740, USA (e-mail: clarke_bp@virginia.edu).

Andrew S. Grimshaw is with the Dept. of Computer Science at the University of Virginia, Charlottesville, VA 22904-4740, USA (e-mail: grimshaw@virginia.edu).

The Legion Grid Portal provides an architecture for integrating a number of existing technologies under a common interface. Although the portal currently uses Legion as the underlying grid infrastructure, it can employ Globus [11] and other grid systems as well. In addition, the portal can employ legacy systems, such as databases, in order to provide users with greater functionality. In this paper, we discuss how the portal uses an off-the-shelf, commodity database for accounting. Moreover, the portal can support application-specific interfaces, called *specific portals*. We describe how users can access a molecular modelling package using a specific portal.

An important benefit of the Legion Grid Portal is that it does not require downloading any of the Legion software on a user's machine. Since the portal operates entirely on the web server, the client machine requires merely a browser installed on it.

Future work in the Legion Grid Portal involves constructing an increasing number of specific portals, providing greater support for grid administrators, exploring interoperability between grid infrastructures, providing a programming interface to grids, and providing increasing support to grid users in the form of superschedulers, information services, interfaces for parameter-space studies, etc.

II. Architecture

The architecture of the Legion Grid Portal is shown in Fig. 1. The architecture is layered, with the highest layer consisting of the user and portal interfaces, the middle layer consisting of the portal implementation along with state information, and the lowest layer consisting of the underlying system in terms of grid infrastructures, legacy systems as well as specific portals. The component identifiers, C1-C6 are explained in Section III, where we explain how we implemented the shaded boxes as well as discuss the relationships between the components.

II.A Grid Software/Services on which the Portal depends

Currently, the Legion Grid Portal depends on the Legion infrastructure for managing a grid. Legion presents an entire grid as a single virtual machine to users [12]. As part of this philosophy, Legion provides a truly distributed file system. This file system is similar to a Unix or Windows file system in terms of command-line or programmatic access, but different from them in terms of the manner in which its components are located, migrated, replicated and retrieved. Moreover, a user's view of a distributed file system is the same no matter which machine he uses to log on to a grid. The contents of a distributed file system are *objects*, a term used to describe any first-class entity in Legion, such as files, directories, machines, disks, users, consoles, programs, etc. The single-machine view of the grid is particularly attractive to the Legion Grid Portal

because it enables presenting a complex environment in a manner familiar to most users.

The portal accesses a grid through Legion's command-line tools. Most of these tools are analogues of Unix tools, but targetted towards the distributed file system of the grid. For example, a tool called `legion_ls` lists the files in a directory of a distributed file system just as `ls` lists the files in a directory of a Unix file system. Likewise, `legion_cat` prints the contents of a file in a distributed file system much as `cat` prints the contents of a file in a Unix file system. For the sake of differentiation, directories in a distributed file system are called *contexts* and a distributed file system itself is called a *context space* [15]. Some Legion tools have no Unix analogues. For example, a tool called `legion_get_acl` retrieves the permissions of an object. Likewise, `legion_list_attributes` retrieves any metadata associated with an object. Currently, Legion's command-line tools are essential for any and all functioning of the portal. A full listing of all Legion commands along with their usage and description is included in the Legion Manuals and man pages that are part of each installation [8].

Most user interactions with the portal involve invoking a command-line tool to perform the task requested by the user. For example, when a user logs in to a grid *via* the portal, she is presented with the listing of the contents of her home directory in the distributed file system for that grid. This listing is procured by running the tool `legion_ls` on her home context. The user could then click on the name of any file or context in her home context. If she clicks on a context, the portal performs a `legion_ls` on the new context. If she clicks on a file, the portal performs a `legion_cat` on that file. The results of either command are parsed and presented to her in an intuitive manner. For any member of a context, the user may request other actions, for example, a listing of the permissions, a listing of any metadata associated with it or its physical location. Moreover, the user can traverse context space similar to what she would do in say, Windows Explorer [18], with the caveat that her access to an object is restricted by her permissions for that object. In addition to traversing context space, a user logged on to a grid *via* the portal can submit jobs as well. Jobs can be submitted through a general interface or through specific portals.

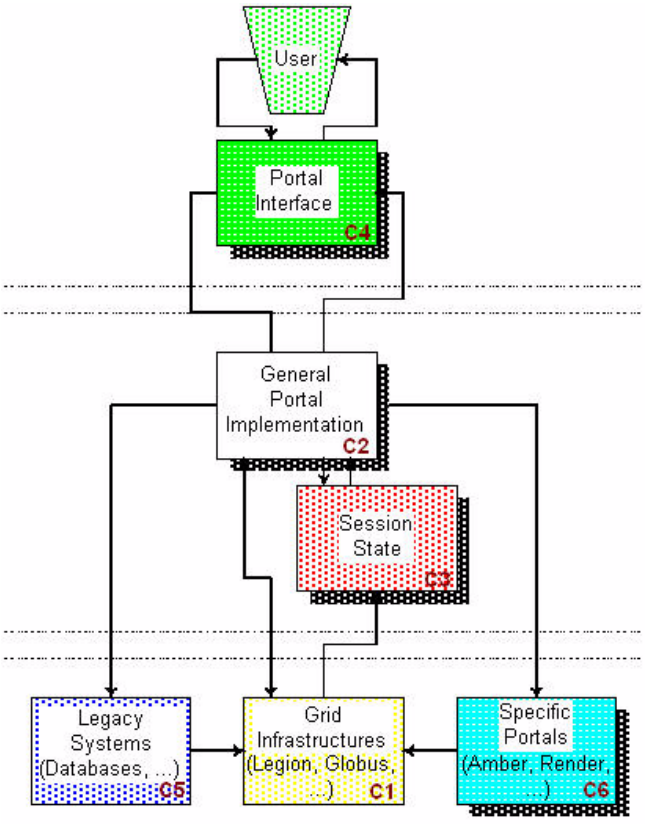

Fig. 1. Architecture of the Legion Grid Portal

The Legion Grid Portal is a convenient interface that masks the tremendous complexity of a grid from a lay user. The portal provides a simple and intuitive interface to a grid that lets the user ignore details about syntax, parameters, permissions, etc. The portal is an attractive tool for introducing users to a grid and giving them a broad overview of its scope without overwhelming them with too much detail. In its current form it is very useful to novice users but less so to advanced users. As we develop more specialised portals, advanced users will benefit in terms of running applications as well being able to manage a grid better.

II.B Grid Software/Services the Portal could use

A large number of grid management tools provided by Legion are not accessible from the portal. These tools are invaluable to grid administrators for maintaining and monitoring a grid. Although these tools are accessible from the command line, they have not yet been incorporated into the portal primarily because most of them require grid administrator privileges. As such, they are not useful to ordinary users who should not be misguided into trying them when they cannot use them. Therefore, an administrator would require different interfaces from ordinary users. However, currently, administrators are not treated any differently from ordinary users in the portal. We are considering increasing the security provided to administrators logging in to a grid. In addition to grid management tools, users and administrators alike may benefit from logs about specific objects. Currently, these logs are not visible from the portal.

A grid managed by Legion can be configured in many different ways, often during run-time. Administrators should be able to exploit this flexibility without resorting to traditional interfaces like command-line tools. They should be able to conduct detailed investigations from the portal and should be able to locate any and all problems from the portal. In addition, creating new objects or services should be simple.

Legion provides other interfaces to a grid in addition to the portal. For example, Legion provides a graphical interface for sharing a Windows directory with other users securely, using grid-level permissions from the context space of a grid. Currently, such tools are stand-alone. Integrating them with the

portal would let users access context space using the tools most convenient for their needs.

II.C Grid Software/Services the Portal requires but not supported by the Grid

Although not envisioned for the near future, a large number of services and software would be attractive if incorporated into the Legion Grid Portal. For example, currently, users cannot take advantage of system services such as Network Weather Service (NWS) or protocols such as Lightweight Directory Access Protocol (LDAP) [14]. Users are not provided with high-level tools, for example, engines to search documents in the distributed file system, or graphing tools for viewing the entire file system at a glance.

II.D Software/Services the Portal uses/requires outside the scope of the Grid

An important task for future work in the Legion Grid Portal is developing specific portals. In particular, we would like to incorporate tools and techniques for application users to view the progress of their runs. For example, currently a user issuing a run of a molecular modelling package such as Amber run can observe the molecule under study periodically. In order to do so, the portal retrieves intermediate files from the run (using a Legion tool called `legion_probe_run`), processes them and creates a protein database (PDB) file using a non-Legion tool called `ambpdb`. This tool is specific to this kind of application. As we develop more specific portals we expect to use an increasing number of such application-specific tools in order to let users view runs.

III. Implementation

The Legion Grid Portal consists of six main components, as shown in Fig. 2. The central component both in the figure and in the design is the General Portal Implementation (C2), which is a Perl CGI script used to process most requests by users. This script is accessed by Portal Interface (C4), which includes the entry page for the portal, subsequent pages generated by the portal and the user, who initiates all actions in the portal. During normal execution, the portal generates caches and session information that are used for authentication as well as speedier execution. These caches and session files, as well as images and logs accessed by the portal are the Session State (C3). Currently, the Legacy System (C5) used in the portal is a commodity database (see Section III.A) along with the scripts necessary to access it. Specific Portals (C6) are used to run specific applications from the portal; since the mechanisms for running specific applications are similar to those for running any application in Legion, this component includes tools, software and scripts for running specific as well as general applications from the portal. Both of these components, C5 and C6 are intricate enough to merit description. The Grid Infrastructure (C1) refers to the services and tools provided in this case by Legion for managing a grid. The list of components can be augmented in order to provide additional functionality to a user. In the subsequent sub-sections, we discuss the existing components. The purpose of these discussions is to present just the general techniques involved.

III.A Commodity Technologies/Software used

The Legion Grid Portal uses Perl [4], PHP [5], MySQL [3] and the Common Gateway Interface (CGI) [7] mechanism for invoking Legion commands. In CGI, the user is presented with a form in which she can fill parameters. Alternatively, the user may be presented with a link with the relevant parameters enumerated. In either case, when the user submits the request, a CGI program on the web server parses the request, retrieves the parameters and executes the appropriate commands. In the case of the Legion Grid Portal, the CGI program on the web server is a Perl script. Part of the portal functionality is implemented in PHP, e.g., accessing accounting and job databases stored in MySQL tables. We show the PHP scripts as part of the component C5 because these scripts were written explicitly for the MySQL legacy system. Likewise, scripts written for specific portals are shown in C6 because they were written for specific visualisation tools. The entire portal is implemented on a Unix operating system. Implementing the portal on a Windows system is part of future work.

III.B Proprietary Technologies/Software developed that can be shared with others

Source code for the Legion Grid Portal is available to Legion users. The algorithms and techniques within the portal are based on standard programming practices found in public documentation [10] [20].

III.C Implementation Details

In this section, we describe implementation details of the Legion Grid Portal at an abstract level. Our intent is to highlight some design decisions as well as present solutions to problems that occurred during design.

III.C.1 **Grid Infrastructure (C1).** The underlying grid infrastructure for the portal is Legion. Legion provides programmatic as well as command-line interfaces to access a grid. However, in the Legion Grid Portal, we take advantage of the command-line interfaces only. The underlying grid infrastructure software for the portal can be changed to any grid system as sophisticated as Legion. For example, the portal can be made to operate on top of Globus. Since Globus does not have a distributed file system, some interfaces present in the portal would have to be discarded. However, significant subsets of the portal, e.g., C3-C6, would require little or no modification.

III.C.2 **General Portal Implementation (C2).** The primary *rôle* of component C2 is to issue Legion commands on behalf of the user. The portal is implemented as a Perl CGI script used to process most of the user's requests. This script has three requirements. Let SERVER denote the machine running the web server, USER denote the user ID owning the script on

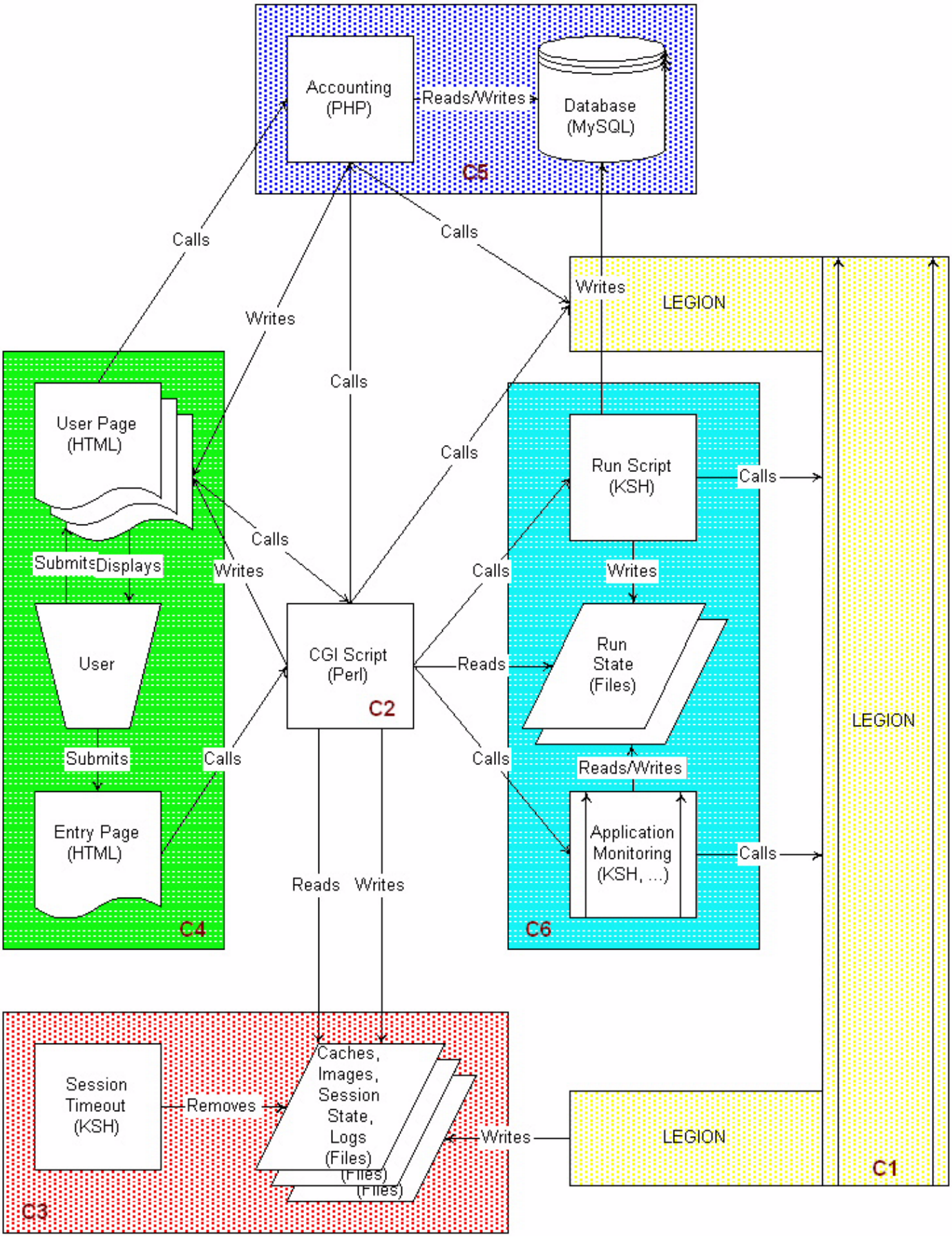

Fig. 2. Details of Components of the Legion Grid Portal

SERVER, and NET denote the name of the grid which the user chooses to access.

1. The directory of USER, for example, /home/USER, must be accessible from SERVER.
2. The directory of the Legion tree, for example, /home/NET, must be accessible from SERVER.
3. Legion must be compiled for the architecture of SERVER and the grid NET must be started. SERVER need not be a Legion host.

If #1 is violated, the user's browser cannot locate the script. If #2 or #3 is violated, every Legion command fails and the grid is inaccessible. Assuming familiarity with Perl, CGI and basic Legion commands, the major steps involved in the General Portal Implementation are:

1. Parse the arguments.
2. Set up the Legion environment and global variables.
3. Get credentials and session id, perhaps by logging in.
4. Generate the context tree for browsing.
5. Select the appropriate handler for the command to run.
6. Check if all arguments are present for the command. If not, generate the page to get the desired arguments and go to step #9.
7. Run the command, displaying its status along the way.
8. Display the output and error of the command.
9. Show the generated page to the user.

Since the interfaces between the Legion Grid Portal (specifically, C2) and Legion are command-line tools, the steps above require little detailed knowledge about Legion. Information about the command-line interfaces are available in the Legion documentation [8]. Step #7 is performed by a handler for the appropriate command. A *handler* is a module written explicitly for checking the required parameters and parsing the output and error states of a Legion command. The handler for a command ensures that proper paramaters are specified for every possible use of the command, and all output and error states are captured and parsed appropriately. Most handlers are simple; however, some of them can perform substantially complex tasks. For example, the handler for `legion_cat` creates a download window wherein the contents of the selected Legion file are shown with the appropriate MIME type. Likewise, the handler for `legion_run`, which runs a legacy application, is expectedly complex. Techniques for constructing handlers for additional Legion commands are explained in the documentation for the Legion Grid Portal. Most handlers eventually invoke a Legion command in a standard manner. This manner involves logging at the start of the command, periodically during the execution of the command, and at the termination of the command. Commands terminate normally, or because a user-specified timeout expired. After the command terminates, the handler parses the output and error of the command to display the results on the next web page generated.

III.C.3 **Session State (C3).** The session state component consists of the various files, caches and session information associated with a particular session initiated by a user as well as general logs maintained by the portal.

The session information particular to the current session for a user includes an environment cache, a credentials cache and a session ID. These files are necessary in order to set up the environment for a user. If a user does *not* use the portal (thus interacting with Legion *via* its command-line interface) her Legion environment can be set up by sourcing a setup file called `setup.sh` or `setup.csh` provided with all Legion installations, and then using `legion_login` to generate the user's credentials. The user's credentials are generally stored in a well-known file protected by the underlying operating system's mechanisms. The user's environment is valid as long as she continues to use the same terminal from which she logged in. However, in the portal, because of the manner in which CGI scripts are executed, each time the user executes a new command, she gets a new terminal. Since requiring the user to log in for every command would be inconvenient, in the portal we manage the environment explicitly. When the user logs in from the portal, we source the setup script provided by Legion and log the user into the grid (using `legion_login` to generate the user's credentials from her authentication object). After a successful login, we cache copies of the user's environment variables as well as her credentials (similar to the MyProxy mechanism in Globus [17]). When the user issues a command, we re-set the environment from the cached copy and re-create the credentials by copying the cached credentials to the appropriate well-known file. The effect is similar to the user's having logged in anew except that the user is unaware that it happened and no additional Legion command is executed. This management of the Legion user's credentials is a security risk only if the Unix user running the CGI script, namely, USER or `nobody`, cannot be trusted.

Access to any Legion functionality, including the credentials file is controlled by a session ID generated by the CGI script. The session ID is a large numeric sequence of random numbers that is extremely hard to forge. For example, in the current implementation, the session ID is constructed by concatenating three random floating-point numbers between 0 and 1000 with 12-digit (decimal) precision. The resulting session ID is a sequence of 39-45 decimal digits with three periods. The probability of an intruder reconstructing a session ID is extremely low, 10^{-45} to be precise. When the user logs in, she is provided with a session ID that is valid as long as she continues to interact with the portal. All requests from the browser are encrypted. The session ID is saved in a file and propagated between consecutive requests from the same browser. If at any time, the session ID from the browser does not match the saved session ID, the session is terminated. Thus, a user's session can be compromised only if he communicates his session ID to an intruder *and* keeps the session valid by interacting with the portal.

The portal maintains a single log that contains timing information about every command issued through the portal. When a Legion command is invoked from the CGI script, its

status is logged periodically as well as when the command starts and ends. The statuses of commands are shown in a status window that a user may choose to view or ignore. By default, the status window is displayed to the user to give him feedback about the execution of the command.

Additionally, a user may choose to view the output and error of every command in separate windows. The portal permits viewing these windows in which the output and error are presented directly from the Legion command, i.e., unprocessed in any manner. The main window continues to show the processed results of the same commands. The outputs and

Fig. 3. Entry Page

errors of most commands issued in a session are stored in separate files. If a user chooses to view the output and error windows for his session and uses a browser's navigation buttons to view previous and next pages, the saved output and error files are retrieved and presented correctly. Moreover, if a user re-issues a previous command, the output and error of that command may be retrieved from the saved files, which thus constitute a cache. Retrieving output and error from caches avoids invoking Legion commands, which can be slow. Since the passage of time as well as a user's actions may invalidate a cache, we invalidate caches aggressively. Caches can be invalidated explicitly by the user, from within the CGI script and periodically by an external session timeout mechanism.

Cached files can be removed periodically by using the Unix `crontab` tool after a session timeout. A `crontab` line similar to the one below is used to check session files every hour and remove those that have not been accessed recently.

```
0 * * * * browser_timeout
```

In this manner, the cached output and error of previous commands can be purged. The purging may remove cached credentials and session IDs as well, thus ensuring that users who forget to log out from the portal are not compromised after a timeout has elapsed. Files associated with runs initiated by users are not purged unless a user explicitly requests to do so. Thus, users may initiate runs and logout or timeout, but continue their runs uninterrupted. The results of the runs can be accessed by logging in again.

Purging caches requires Unix permissions to delete the associated files. Therefore, we recommend running the CGI script with `cgiwrap` [16]. If the script is run without `cgiwrap`, it runs as some special user, usually `nobody`. Consequently, only the user `nobody` is permitted to remove the cached files. However, on most web server installations, `nobody`'s permissions are restricted to prevent running many Unix commands or even logging in. If the script is run with `cgiwrap`, a non-privileged user, e.g., `USER`, could log in and clean up caches and session state on errors or timeouts. `cgiwrap` operation can be selected or ignored by changing the entry page alone for the portal.

III.C.4 **Portal Interface (C4).** The entry page as well as the user pages generated by the portal constitute its interface. The generated pages contain enough information to invoke the portal subsequently. The portal state of a user's session is passed between consecutive CGI invocations by normal CGI mechanisms. These mechanisms involve either setting hidden widgets in forms or explicitly enumerating the state in links. Alternatively, session cookies can be employed if they are supported by the browser [6]. The CGI script uses seven elements to reconstruct the session state for a user. These elements are: the Legion user name, the Legion grid name, the command to be executed (defaults to `legion_ls`), the current working context, the session ID, the timeout selected by the user and the verbosity level for the portal.

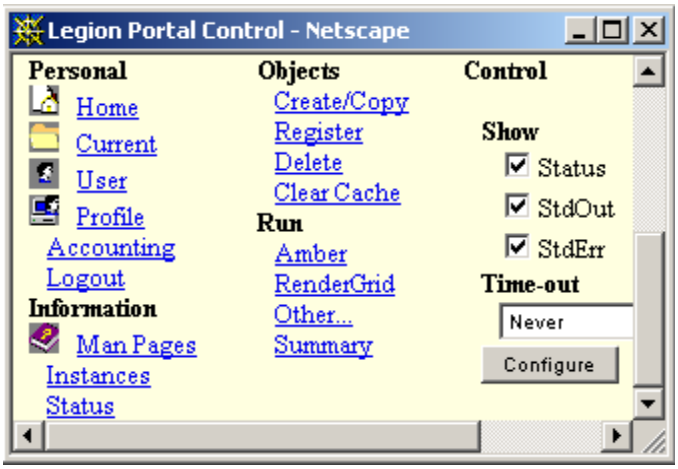

Fig. 5. Control Panel

The entry page (Fig. 3) requests the Legion user name, the Legion password, the Legion grid name, the command to be executed, the timeout and the verbosity level from the user. In turn, it invokes the CGI script (C2), which computes the current context to be the user's home context and generates a session ID. From this point on, every generated page (Fig. 4 onwards) contains the above-mentioned seven elements from which the user's Legion environment can be reconstructed. For any subsequent invocation of the CGI script, if either the Legion user name, the Legion grid name or the session ID are absent or incorrect, the Legion Grid Portal reports an error, terminates the session, and requests the user to re-login.

After a user logs in, the main window of his browser displays pages generated by the portal (Fig. 4). In addition to this window, the portal may open up to five windows on behalf of the user. First, a control window is always opened with a panel of buttons and links that are used frequently (Fig. 5). For example, the control window has links for browsing the user's home context, running an application and copying files between the grid file system provided by Legion and the user's file system provided by the operating system. Second, a status window is opened by default to show the logs of commands (see Section III.C.3 and Fig. 6). Third, an output window may be opened to show the raw output of commands (Fig. 7). Fourth, an error window may be opened to show the raw error of commands (Fig. 8). Opening and closing the status, output and error windows can be accomplished by setting the verbosity level of the portal. Fifth, a download window may be opened with the appropriate mime type if the user decides to view the contents of a file.

III.C.5 **Legacy Systems (C5).** Currently, accounting and job monitoring in the Legion Grid Portal are accomplished by taking advantage of a commodity database system. The portal provides the interfaces to access this legacy system through a PHP [5] script that accesses a MySQL [3] database as well as Legion. The PHP script executes Legion commands to obtain resource consumption data about a user's Legion objects

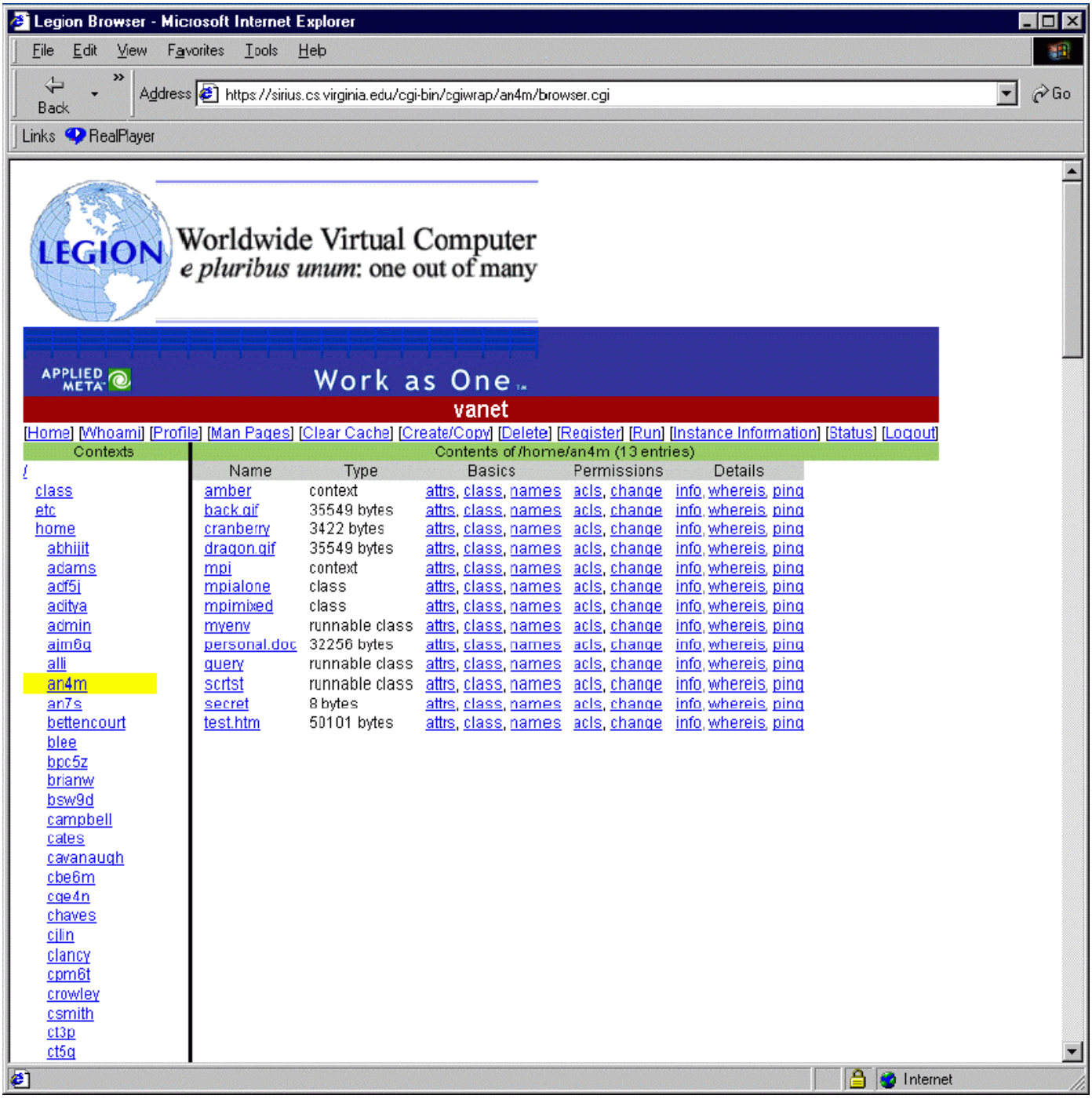

Fig. 4. User Page generated after login

(Fig. 9). The resources contributed by the user towards a grid can also be obtained.

Whenever a user requests accounting information, the portal seamlessly transfers control from C2 to C5. All session information is transferred by CGI mechanisms to the PHP script which manages the information in the same manner as the Perl script in C2. Subsequently, when the user returns to non-accounting actions, control is transferred seamlessly from C5 to C2. The user is not necessarily aware that different CGI scripts are used to process different requests because the session state required by the portal is small and can be transferred easily. Our success in adding accounting functionality to the Legion Grid Portal by way of transfer of control between these scripts is encouraging because it implies that we can continue to increase the functionality provided through the portal in this manner. Moreover, the same mechanisms used to transfer control between C2 and C5 can be used to transfer control between the Legion Grid Portal and other grid portals such as GridPort [1]. Thus, the portal can be extended to provide desirable interoperability between various grid infrastructures such as Legion and Globus.

III.C.6 **Specific Portals (C6).** A significant task enabled by the Legion Grid Portal is starting up runs on behalf of a user. A

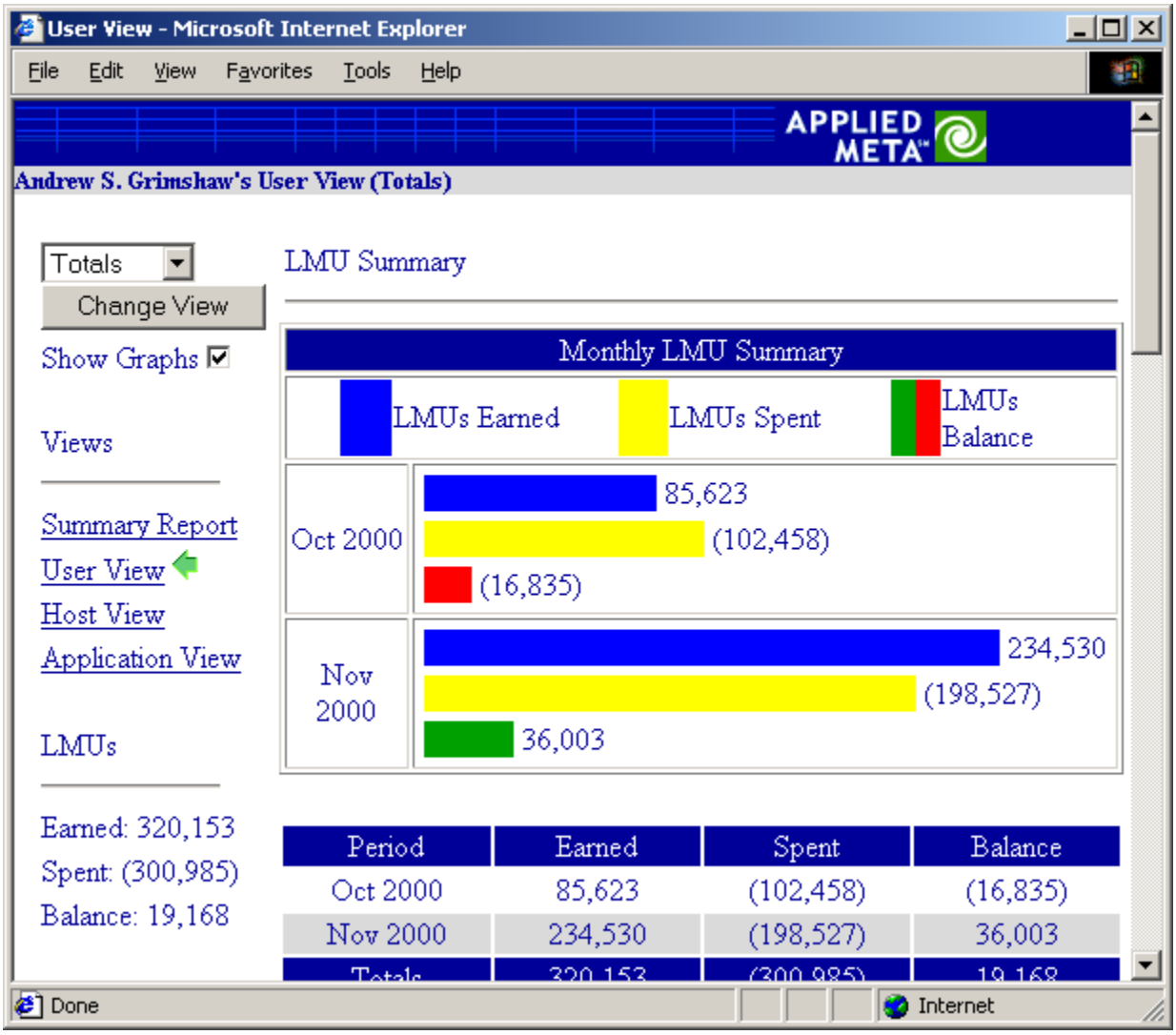

Fig. 9. Accounting Information

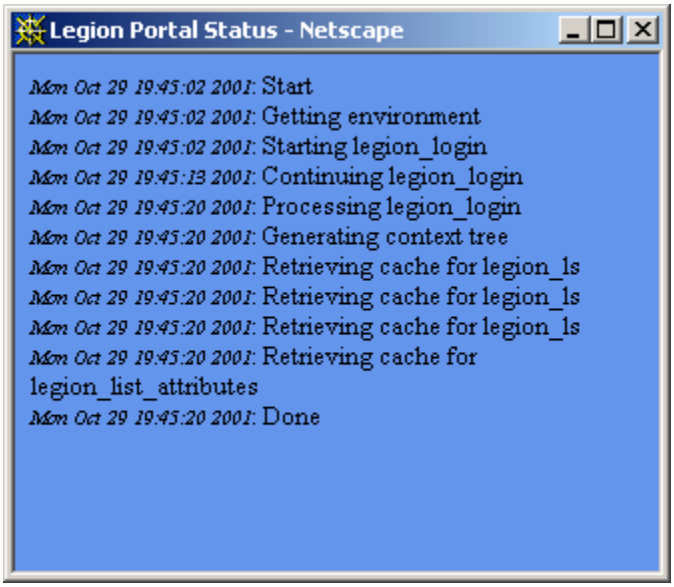

Fig. 6. Status Window

command like `legion_run` is much more complex than ordinary Legion commands. Moreover, since runs may execute

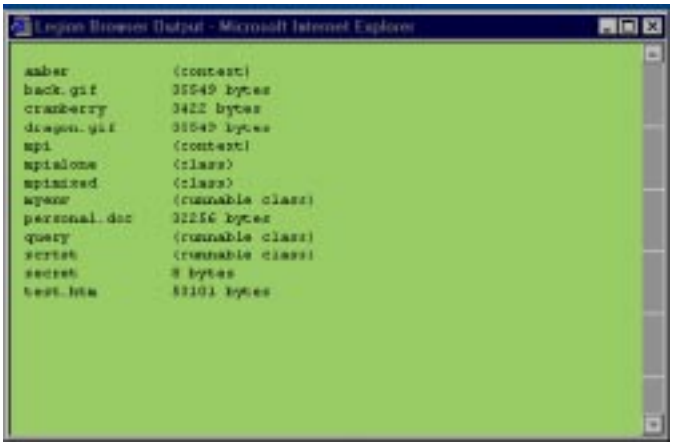

Fig. 7. Output Window

for a long time, the user's browser cannot be made to wait until the command is complete. This asynchronicity makes the handlers for executing runs complex.

Before starting a run, the grid portal assembles the arguments, parameters and input files for the run. A handler for a run command provides means for the user to input all

arguments. Therefore, it generates buttons, boxes and widgets for arguments for the run itself and arguments to the Legion run command (Fig. 10). Necessary arguments are checked for existence and sanity. Reasonable defaults are chosen for the remaining arguments. If the run requires Unix/Windows input files, the browser can send their contents in a multipart form (such input files cannot exceed around 5MB in size; this limitation is imposed by the HTTP protocol for multipart forms). These files are saved on the web server in a sub-directory specially created for that run.

The run script invokes Legion commands to initiate the run. Along with initiating the run, the script records information about the run in the database in C5. A user can access the status of her runs from the page generated by the portal for the user immediately after the run begins. (Fig. 11, Fig. 12, Fig. 13). Additionally, a user browsing her accounting information can access the status of her runs from the database using interfaces provided in C5 (Fig. 14). The user can monitor these pages periodically to view the results of the run as it progresses. From these page, the user can record the remote machine on which the run executes, the working directory of the run and any other

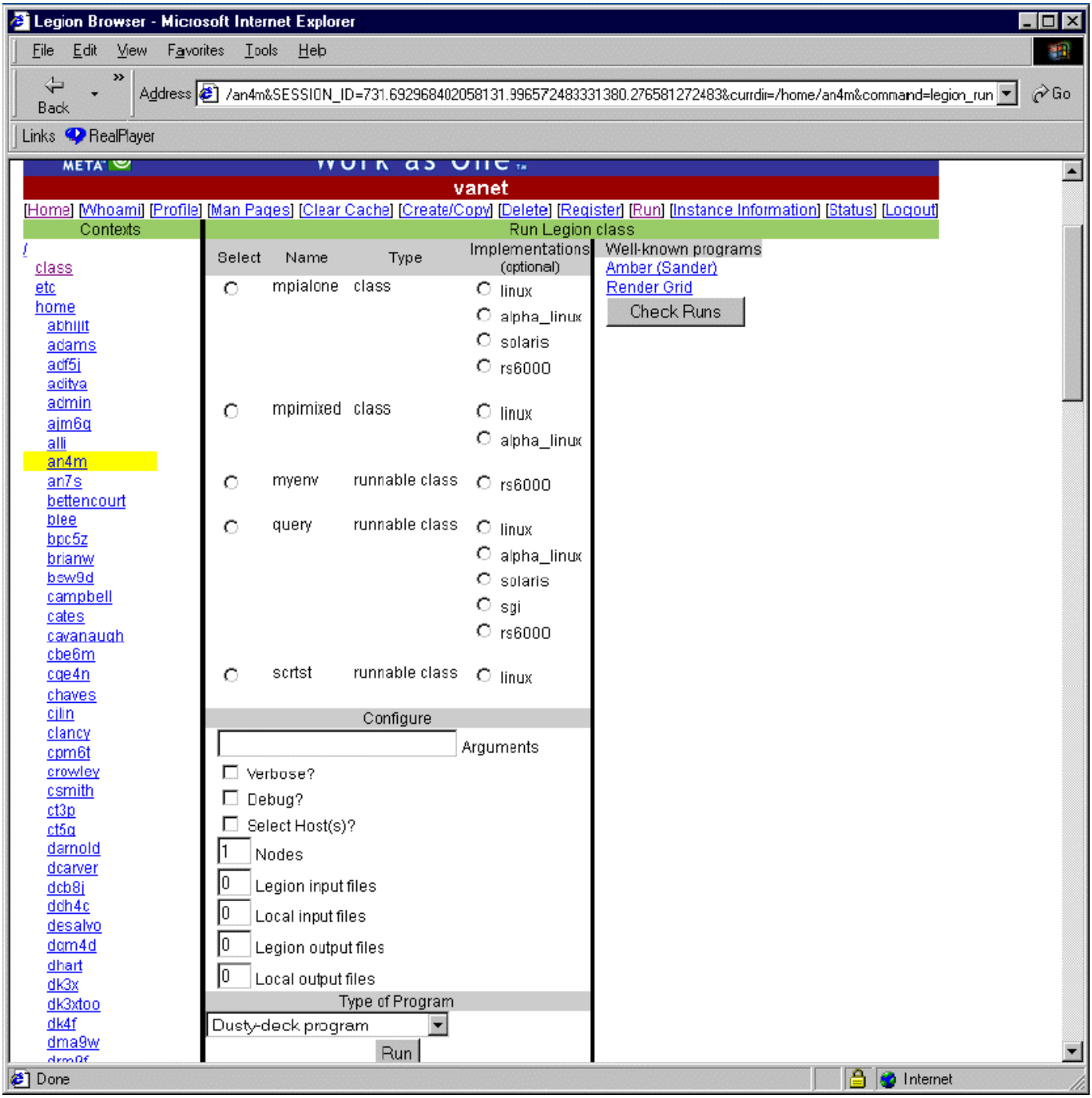

Fig. 10. Initiating a Run

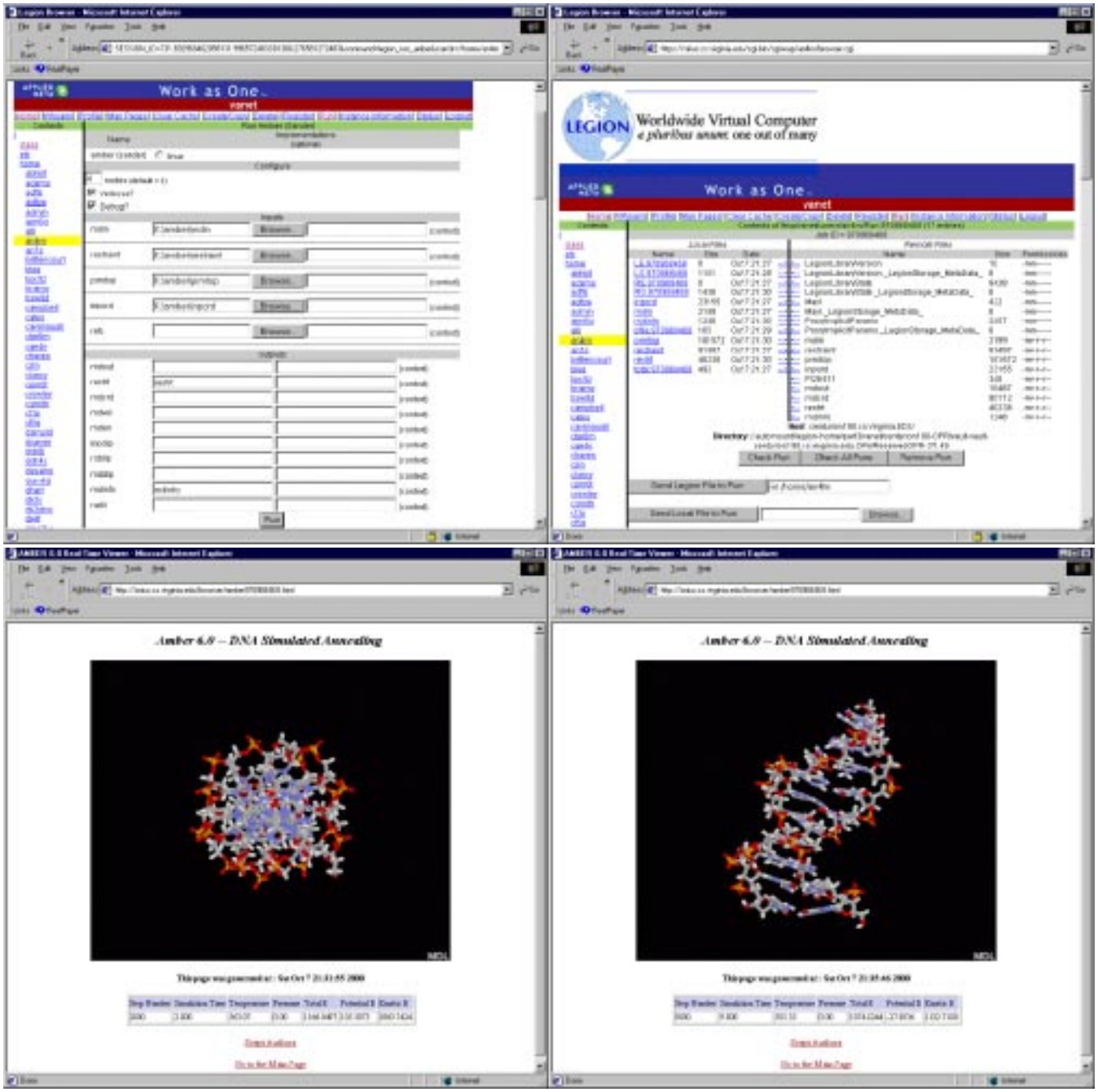

Fig. 11. Specific Portal for Amber

information Legion provides about the run. In addition, he can transfer intermediate files from the remote machine to the server (and thence to his own machine) as well as transfer files from his own machine to the remote machine (*via* the web server). Transferring intermediate files out from the run is useful for viewing the run periodically as well as checkpointing. Transferring files in to the run is useful for computational steering.

The run script can be used to initiate runs of well-known applications, i.e., applications for which the binaries are registered under well-known and widely-accessible runnable classes [15]. For example, we have developed a portal for a molecular modelling package called Amber and an astronomical modelling package called Hawley-Hydro. The portals have application-specific widgets that let users enter input files intuitively (Fig. 11, Fig. 12). However, from the Legion Grid Portal's perspective, these applications are identical to any other application. In other words, the mechanisms to initiate an Amber or Hawley-Hydro run are identical to those for any other run. After the run is initiated, the portal provides an additional viewing capability for the run. This additional view for Amber requires a plug-in called Chime [2] to be installed on the user's browser. With this plug-in, the user can view the progress of the run graphically in addition to the usual view provided by Legion. Since specific portals are merely more convenient interfaces to the basic

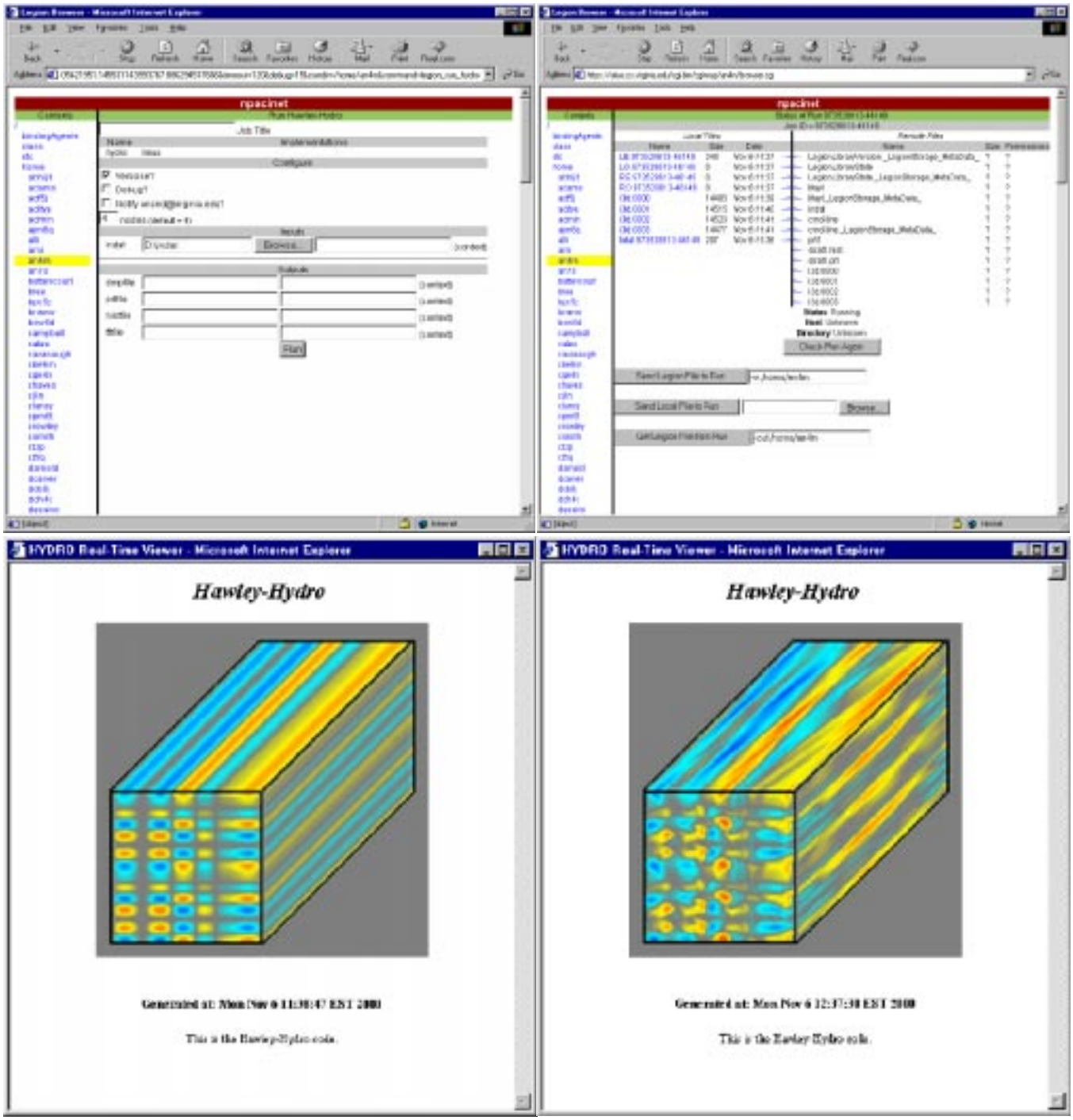

Fig. 12. Specific Portal for Hawley-Hydro

functionality of initiating a run, creating new portals is simple. For example, a specific portal for CHARMM [9] should take only a few hours to construct once the particulars of the application are available.

IV. Supported Grid Services

The Legion Grid Portal supports a significant subset of the capabilities and features of a grid system using Legion; the remaining can be added with a small amount of effort.

IV.A Security

Security in the Legion Grid Portal is addressed in a number of ways. The portal takes advantage of the security infrastructure provided by Legion. In Legion, typically users log on to a grid using a login-password combination. Legion generates credentials for the user to be used for that session. Currently, the credentials are based on public-private key pairs. The method for generating the credentials is irrelevant to the portal. However, the portal manages a persistent copy of the credentials on behalf of the user.

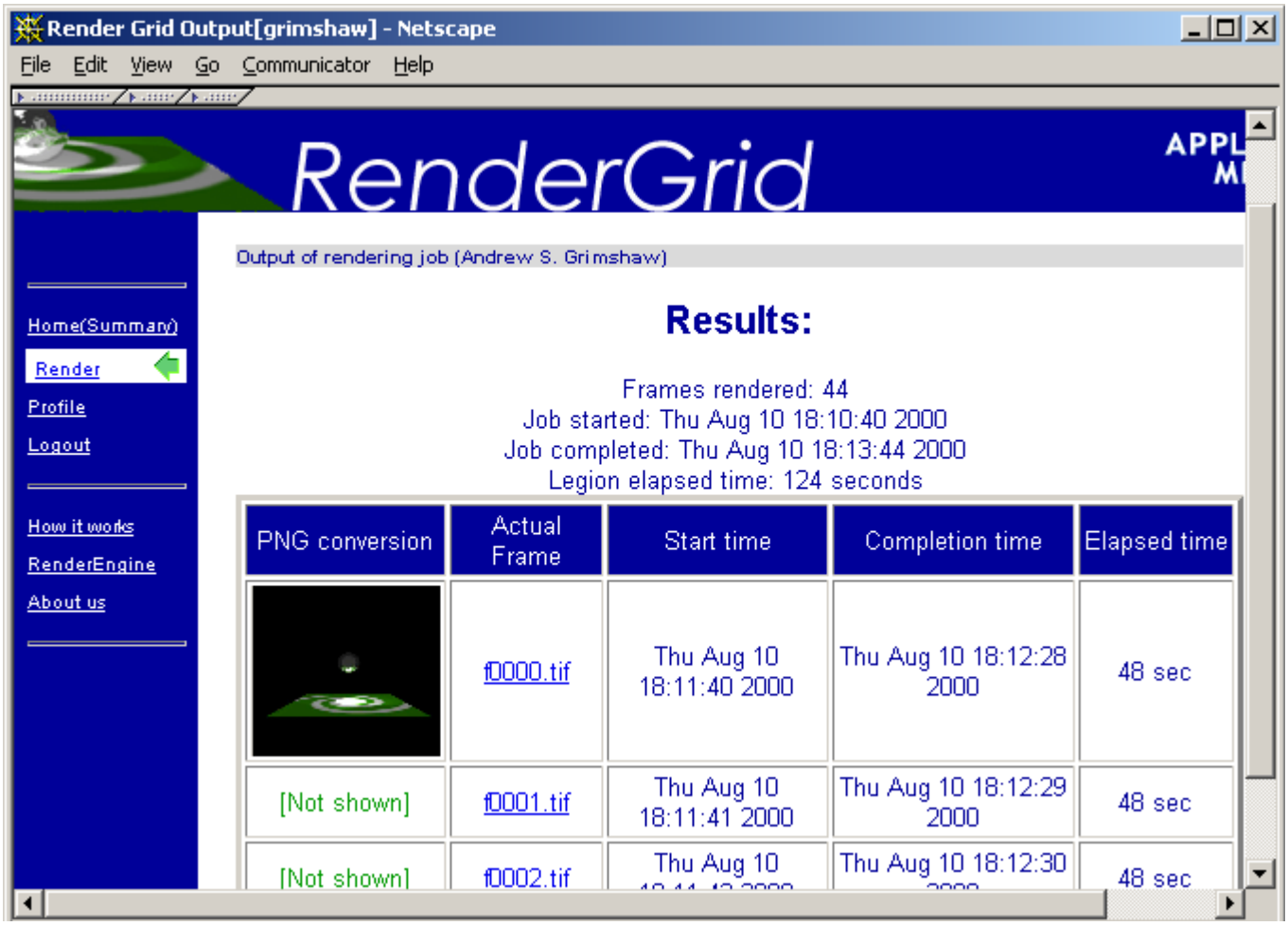

Fig. 13. Specific Portal for Rendering

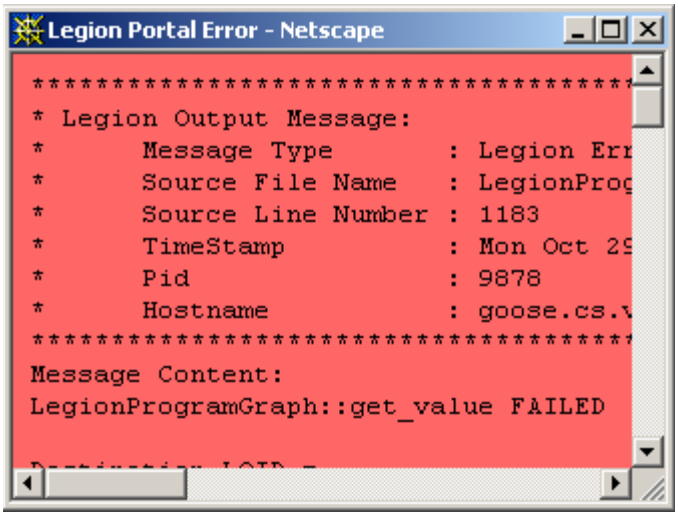

Fig. 8. Error Window

When a user logs on to a grid from the command-line, Legion queries his login ID and password. If the pair is valid, Legion generates a file which stores the user's credentials. The file is named based on the process ID of the user's terminal; if the user opens another terminal, the credentials file is not valid for the new terminal. However, in the portal, subsequent commands are executed in different shells. Since we cannot the expect the user to supply an ID and password for every command, and since we prefer not to store the user name and password either on a disk on the web server or in the session state of the browser, we manage the credentials explicitly. Access to the managed credentials are moderated with a session ID generated for every session (see Section III.C.3). The session ID is hard, if not impossible, to duplicate. Consequently, a user can access only his credentials. Moreover, if the user is inactive for some time (currently, two hours), the session ID and the saved credentials file are both invalidated. This invalidation requires the user to log in again, but ensures that the user's credentials are erased if he forgets to log out or his browser terminates unexpectedly.

A potential security risk in the portal occurs when the user logs in for the first time. At this time, the `legion_login` command is invoked with the user's Legion login ID and password. A Unix user on the web server could potentially view the password in clear-text by executing a `ps` command exactly during the `legion_login` command. Currently, we avoid this problem by restricting the people permitted to be Unix users on the web server. Another solution which we are considering is to modify the `legion_login` command to take not the clear-text password but the name of an encrypted file as a password parameter. With this solution, an intruder

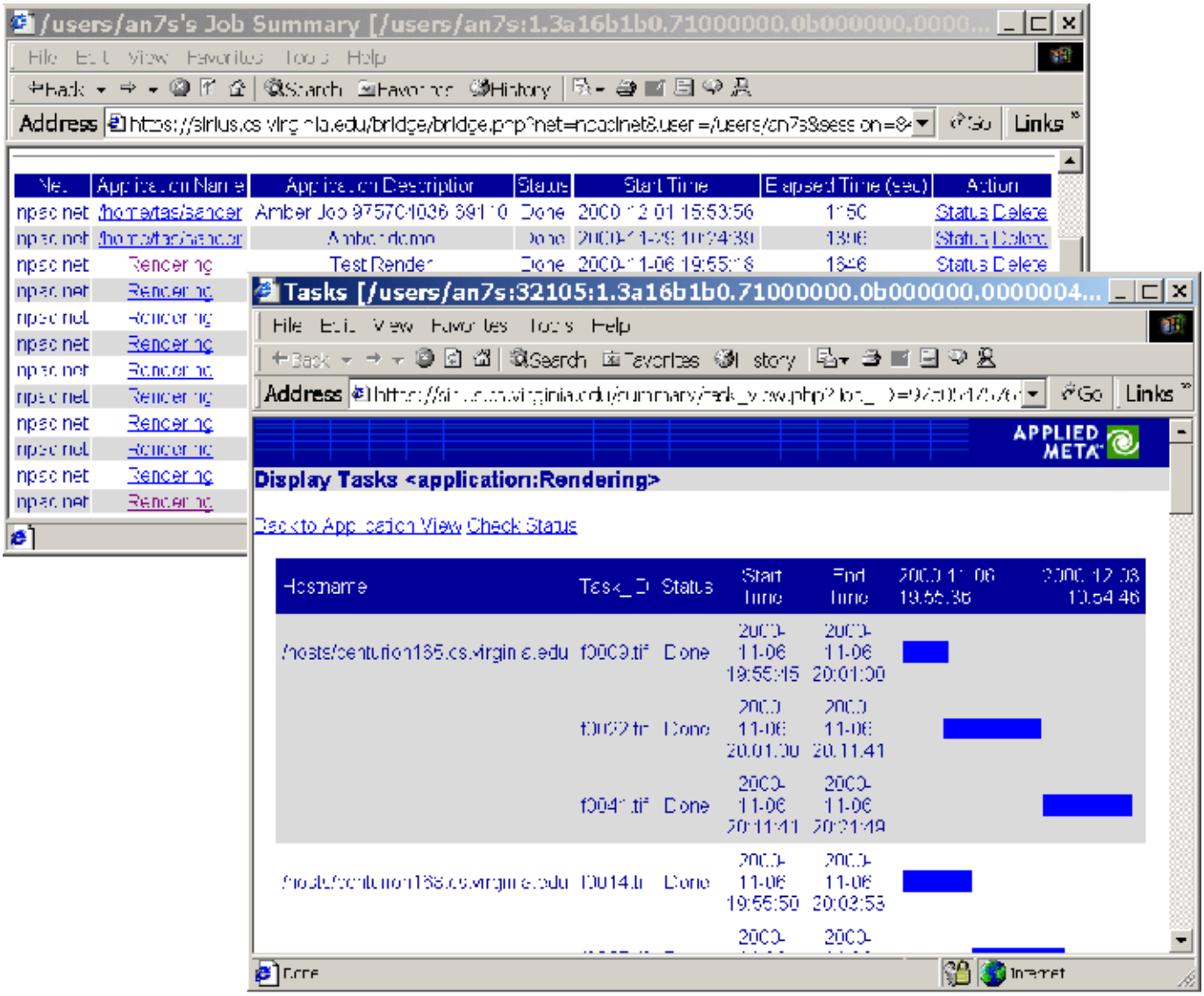

Fig. 14. Status of a Parameter-Space Study

may be able to see the name of the file but may not be able to access the file in any manner. However, neither solution is secure enough if the superuser or web server user herself is the intruder. Since every command issued via the portal executes under the Unix ID of the web server user, this user can access any information pertaining to any Legion user. Likewise, the superuser on the web server can access any information pertaining to any user. We believe that insecure as this situation is, it may be unavoidable. On large installations it is common for privileged users such as the superuser to be able to access any information pertaining to any user. A responsible choice of privileged users and judicious encrypting of critical data may be the only reasonable solutions.

IV.B Information Services

The portal permits accessing all information services that can be accessed by a command-line user. For example, a portal user can browse context space (Fig. 15), view the metadata for any object for which he has permissions to do so (Fig. 16), or view all of the hosts in a grid (Fig. 17, Fig. 18). In Legion, collection objects are repositories of metadata of other objects. For example, a certain frequently-used collection object stores metadata about every host in the grid. This collection is queried during the scheduling process. A command-line user may issue one command to procure the data stored by any collection. Currently, a portal user cannot issue such a command although adding such a functionality to the portal is trivial. However, a portal user can access a collection's data implicitly, for example, during the scheduling process.

IV.C Scheduling

Legion supports explicit and implicit scheduling. In explicit scheduling, a user specifies the host on which she wishes to run. In implicit scheduling, the grid infrastructure selects the host on which the user runs. In Legion, creating any object, whether it be a file or an instance of a program, involves a scheduling process. Typically, users schedule implicitly, especially during the creates of non-programs. The Legion

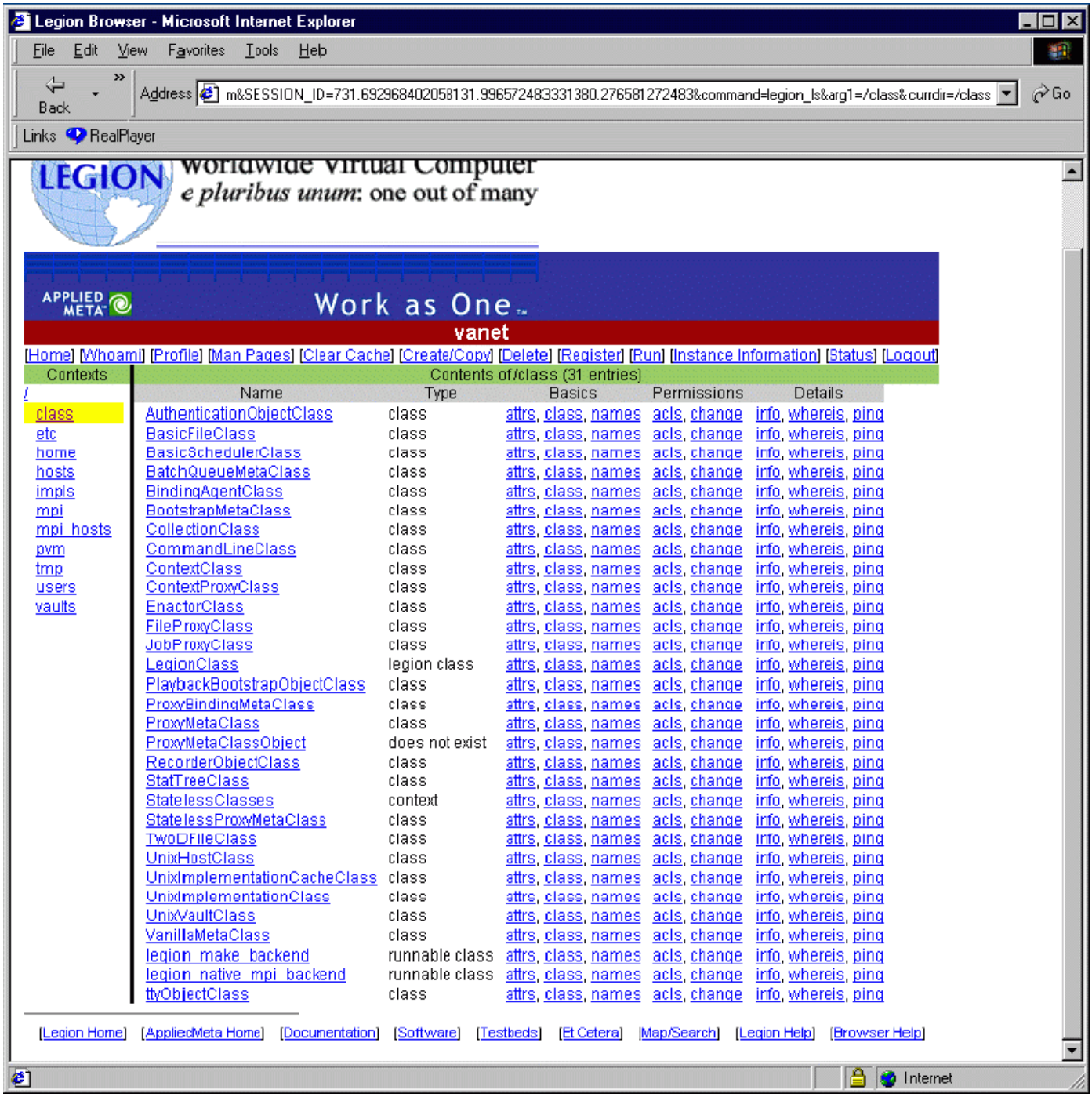

Fig. 15. Browsing Contexts

Grid Portal supports only implicit creates for non-programs, although adding explicit creates is a trivial task. The portal supports both implicit and explicit creates for programs. Therefore, users can choose exactly the resources on which they would like to run their applications. We are investigating mechanisms and interfaces for letting users select sets of hosts for large runs, such as parameter-space studies. Also, we are investigating constructing superschedulers that let users select from a wide variety of resources.

IV.D Data Transfer

The Legion Grid Portal supports automatic data transfer for applications insofar as it is not limited by the CGI mechanism itself. In Legion, the commands for running applications provide switches by which input and output files can be specified either from the local file system or from the grid file system provided by Legion. Moreover, applications may access the grid file system directly, thus obviating the need for any of these switches. The portal supports all of these modes of operation. However, if the input/output switches are specified, certain CGI restrictions become apparent. If the user specifies

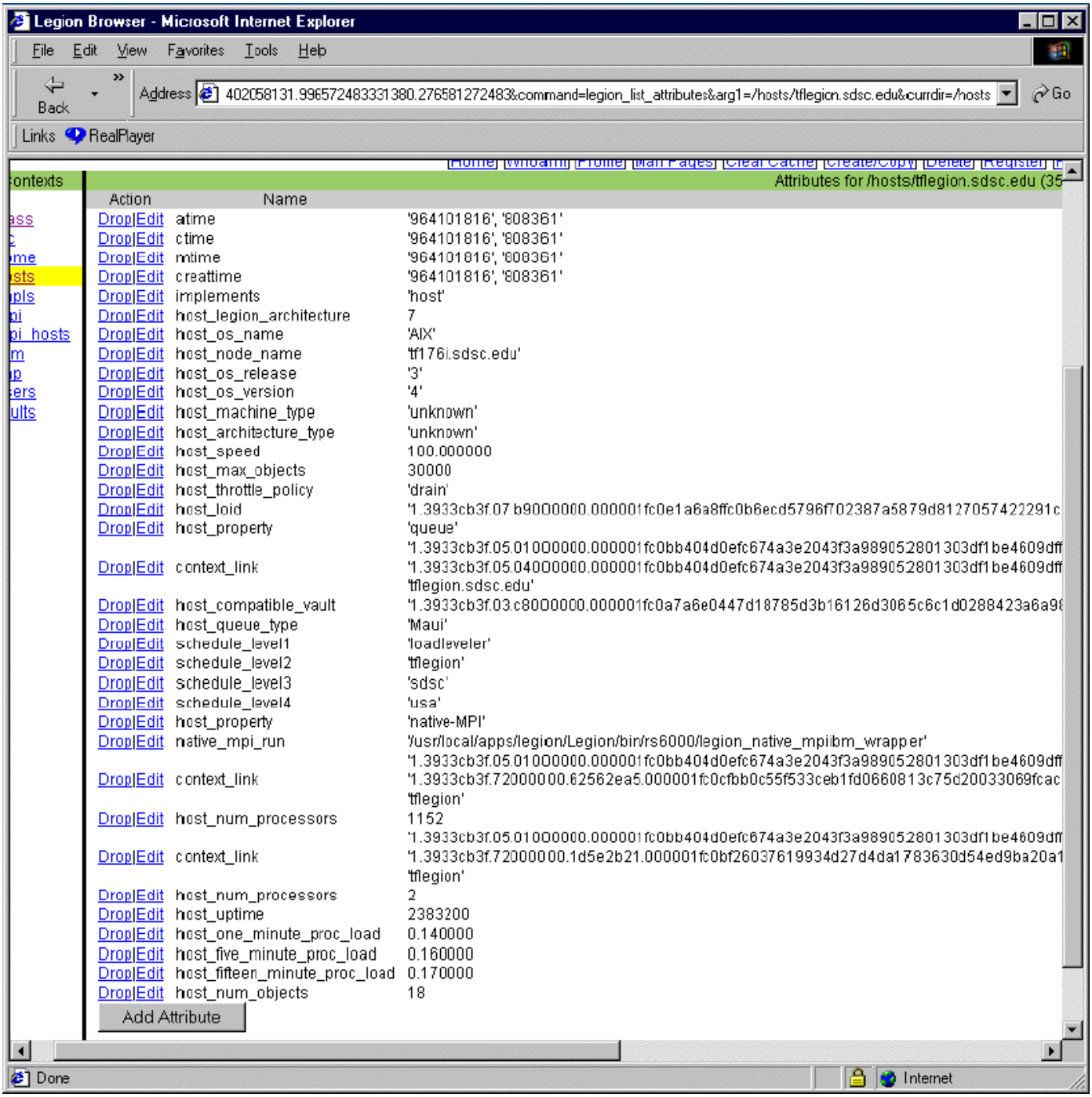

Fig. 16. Metadata for a Host Object

that the input and/or output files are to be accessed from the grid file system, then the portal has no restrictions. If the user specifies that the input files be accessed from the local file system, CGI provides a way to upload the files to the web server from where they can be supplied to the appropriate Legion command. However, CGI imposes a limit on the size of the uploaded file, currently of the order of a few Mbytes. If the user specifies that the output files be stored to the local file system, then there is no mechanism within CGI to store the files automatically because the user must authorise the storage. Therefore, the solution in the portal is to store the file on the web server and provide a mechanism to download it to the user's local file system.

IV.E Additional Grid Services

The Legion Grid Portal provides excellent facilities for monitoring jobs, viewing jobs as they progress, transferring intermediate files and viewing accounting information. The Legion tools for running applications provide means by which the status of a currently-executing job can be viewed. Moreover, the tools provide mechanisms for sending and retrieving intermediate files. The portal enables users to perform these tasks. For some applications, we have

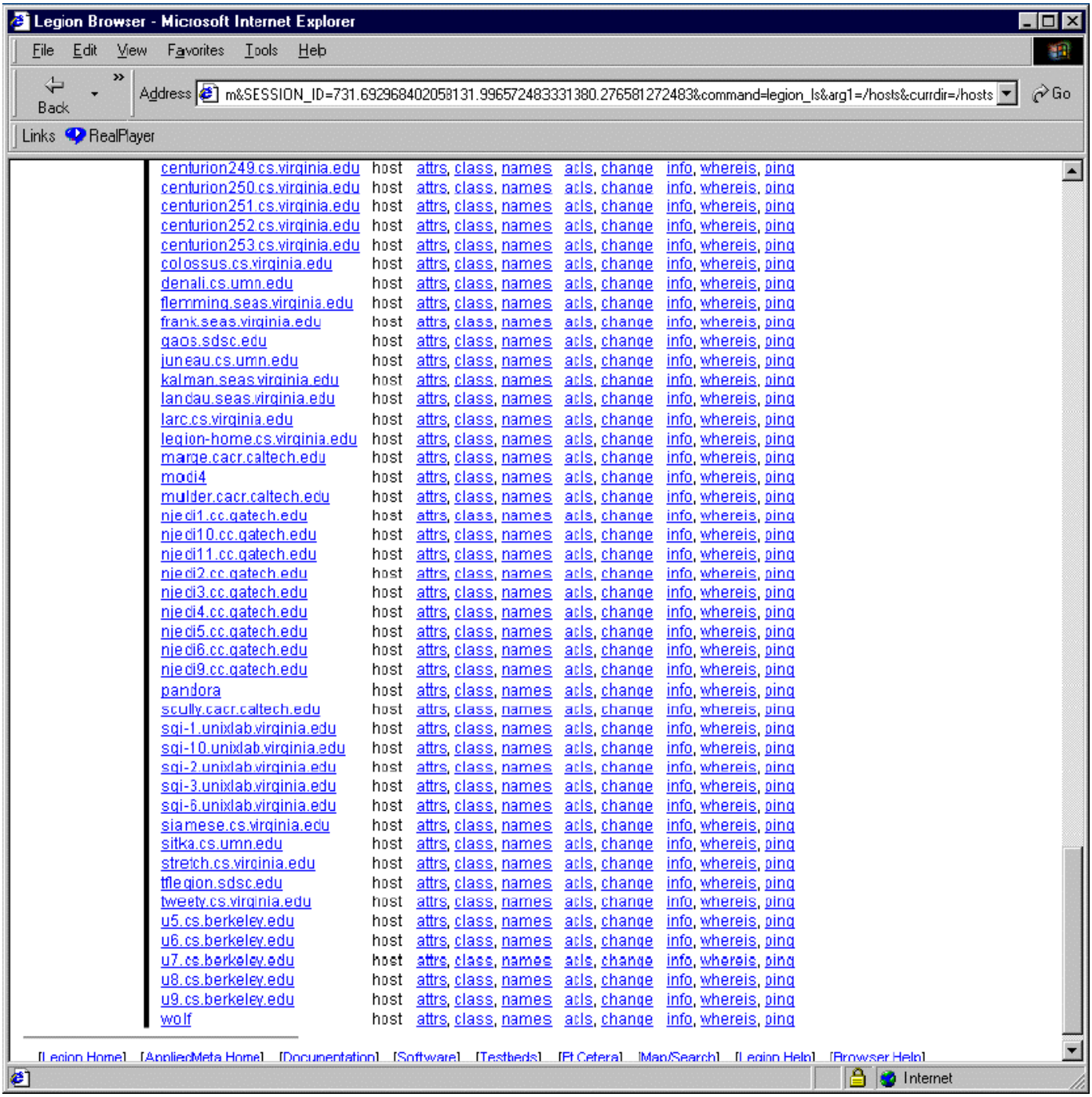

Fig. 17. Viewing Hosts in a Grid

constructed specific portals which have all of the functionality associated with running any application using Legion in addition to specific interfaces for visualising the progress of a job. For example, using the Amber portal users can view the progress of an Amber job graphically. After they submit a job through the Legion Grid Portal, a new window appears in which the intermediate state of the molecule being studied is displayed. The portal displays this view by accessing intermediate files generated by the application periodically and converting them into a protein database file using commodity tools. The protein database file can be viewed graphically using the Chime viewer. Likewise, the portal also permits viewing Hawley-Hydro jobs by accessing intermediate files and converting them into GIF images. Finally, in the case of some parameter-space studies, the portal provides a means of viewing the status of each job in an abbreviated manner, giving the user an aggregate view of the application as a whole. Integrated with the job views are accounting views that show the resource donation and consumption by every grid user.

IV.F New Grid Interfaces arising from the Portal

The original design goal of the Legion Grid Portal was to present an intuitive front-end to the command-line tools

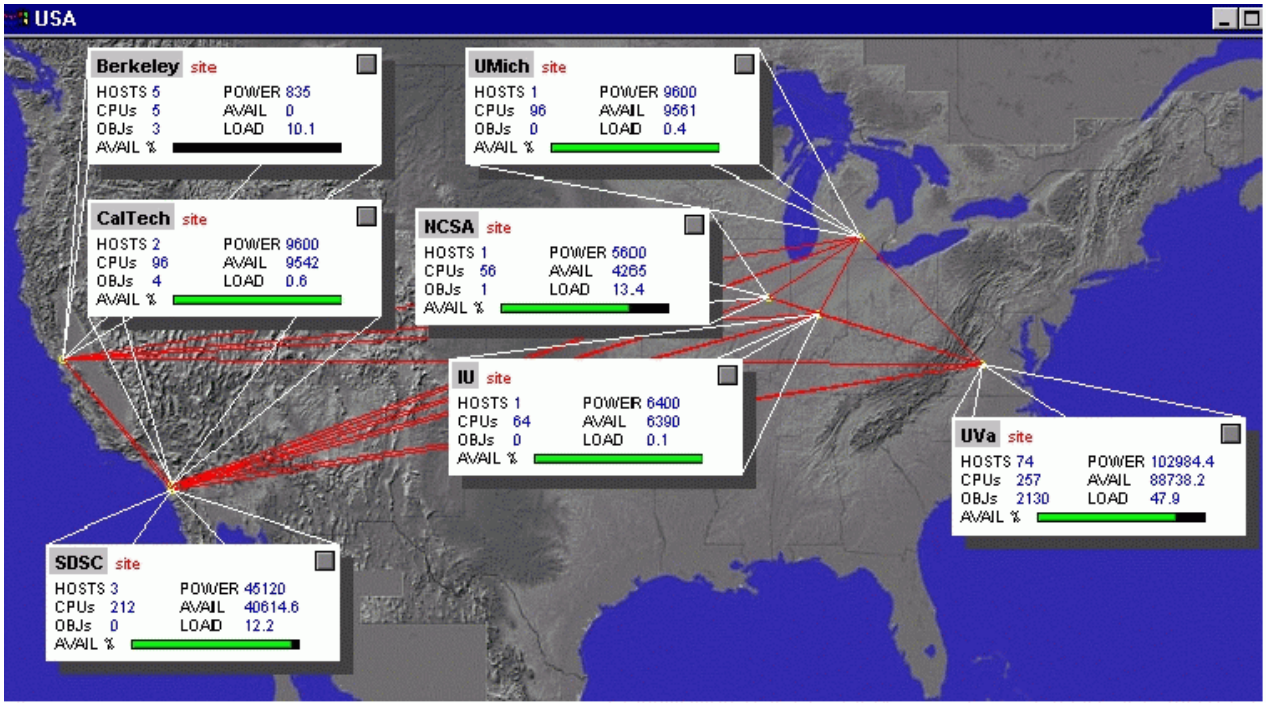

Fig. 18. Worldwide Grid managed by Legion

available to a Legion user. However, as the design of the portal progressed, we discovered that we had to change some design details in Legion. In particular, some of the changes were:

• The error modes of many tools were solidified and standardised. Previously, standards for writing command-line tools were lax; tools reported different error codes for the same error, had different conventions for reporting outputs, had different levels of versbosity, etc. Although standardisation of tools is not yet complete, we are taking steps in that direction. In particular, we have realised that Legion tools must be as robust and standard as the Unix suite of tools, if not more so.

• Non-blocking modes of running applications was developed. Previously, Legion users used to initiate runs in “blocking” mode, i.e., the tool initiating the run would wait until the run completed. Such a mode is highly undesirable in a portal because most browsers will terminate a connection after a period of inactivity. Since we cannot expect runs to periodically output text for a browser, and we cannot expect browsers to sustain connections notwithstanding, we developed tools for running applications in “non-blocking” or asynchronous mode. In this mode, the user (or the portal on behalf of the user) initiates the run and collects a token or ticket that identifies the job from Legion.

• Probing/monitoring runs was developed (with suggestions from existing users). After we developed non-blocking runs, the natural design progression was to enable probing or monitoring runs. Using the token or ticket generated by Legion, we were able to develop tools that report on the progress of the run. Typically, these reports include information about the machine on which the run is executing, the working directory of the run, the names and sizes of the files generated, etc. The ability to probe runs has proven to be extremely helpful to Legion users. In the case of specific portals, the ability to probe runs has enabled the use of visualisation tools that use intermediate files to display the progress of runs.

• A proxy tool was designed to make command-line tools run faster by pre-initiallising the Legion library. In Legion, every command-line tool initiallises a Legion library. When executing multiple tools frequently, as in the case of the portal, repeated initiallisations of the Legion library can represent a large overhead. We developed a proxy tool which can initiallise the Legion library once for an entire session. Multiple tool invocations result in connections to the proxy which can perform the functions of many tools quickly. Preliminary investigations have shown that the speedup in tool execution is around 33%-50%.

• We are re-thinking mechanisms to run general parameter-space studies. By definition, parameter-space studies require large sets of parameter values. Typically, each set of parameters is supplied in one or more files. A user desiring to conduct a parameter-space study must construct the sets of files for each run in the parameter space. Constructing those files is an application-specific task. However, submitting those files for initiating a large number of runs can be complex. For command-line users, Legion provides a tool called `legion_run_multi`, that enables them to specify the sets of files. For portal users, specifying sets of files can be difficult — specifying each file singly is tedious, specifying multiple files with wildcards is not possible because the web server and client have different filesystems, and specifying all the files within one single archive is difficult because of file size limits inherent in CGI transactions. Although it seems like the only option is for users to use Legion’s distributed file system

(which could be accessible from both the server and the client), we are exploring methods by which the user can use her Unix/ Windows file system as well.

V. Project Status and Future Plans

The Legion Grid Portal has been operational since February 2000. Over time, it has acquired an increasing number of features and undergone several changes in its look-and-feel. The portal has been made more robust and more intuitive to the user. The entire design of the portal has been motivated by the desire to present users with an interface to a grid that is not more complicated than a few mouse clicks and occasional typing. Informal studies have shown that users can grasp important grid concepts much more quickly through the portal than with other interfaces. Consequently, we are increasing the usage of the Legion Grid Portal in tutorials.

The tasks that remain for the Legion Grid Portal fall into the following categories:

1. Increasing access to Legion functionality for lay users. Currently, the portal enables users to access only a small albeit critical subset of Legion. We expect that as the portal matures, more and more Legion commands will become accessible from the portal. Moreover, interesting new compositions of Legion tools will become commonplace in the portal.
2. Increasing the number of specific portals. Currently, we have portals for three applications — Amber, Hawley-Hydro and RenderGrid. Such portals are well-suited for introducing high-performance users to grids. The availability of tools for monitoring specific applications would accelerate the development of specific portals in the Legion Grid Portal.
3. Increasing the number of tools for administrative users. Currently, the administrator of a grid is treated as just another user on the portal. The tools available to such a user are identical to the tools available to any user. We expect to add tools that only administrative users can employ. Also, we expect to make log files available to such users. Such an approach will make the management of a grid intuitive and simple to administrators.
4. Providing a programming interface for grids. Legion provides an abstract programming model based on dataflow graphs. This model is attractive to developers of grid services. We expect to provide such developers with tools to construct their services over Legion.
5. Exploring grid interoperability. The portal has the potential to unify high-level functionality provided by different grid infrastructures. We expect to study how the relative strengths of different approaches to grid can be utilised within the common interface provided by the Legion Grid Portal.

Acknowledgement

We thank Mark Morgan at Avaki Corporation for his suggestions on the design and implementation of parts of the Legion Grid Portal, Katherine Holcomb for setting up the web server for the Legion project at the University of Virginia and “Referee 1” for comments that helped us clarify several concepts in this paper as well as correct several errors and inaccuracies.

References

[1] —, “GridPort”, `gridport.npaci.edu`.
[2] —, “MDL Information Systems, Inc.”, `www.mdli.com`.
[3] —, “MySQL”, `www.mysql.com`.
[4] —, “Perl Mongers”, `www.perl.org`.
[5] —, “PHP”, `www.php.net`.
[6] —, “Server-Side JavaScript Guide”, `developer.netscape.com`.
[7] —, “The Common Gateway Interface”, `hoohoo.ncsa.uiuc.edu/cgi`.
[8] —, “The Legion Manuals (v1.7)”, University of Virginia, October 2000.
[9] Brooks, B. R., Bruccoleri, R. E., Olafson, B. D., States, D. J., Swaminathan, S., Karplus, M., “CHARMM: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations”, *J. Comp. Chem.*, vol. 4, 1983.
[10] Christiansen, T., Torkington, N., *Perl Cookbook*, O’Reilly & Associates, ISBN: 1-56592-243-3, 1998.
[11] Foster, I., Kesselman, C., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999.
[12] Grimshaw, A. S., Wulf, W. A., “The Legion Vision of a Worldwide Virtual Computer”, *Comm. of the ACM*, vol. 40, no. 1, January 1997.
[13] Grimshaw, A. S., Ferrari, A. J., Lindahl, G., Holcomb, K., “Metasystems”, *Comm. of the ACM*, vol. 41, no. 11, November 1998.
[14] Howes, T., Smith, M., *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*, Macmillan Technical Publishing, ISBN: 1-57870-000-0, 1997.
[15] Natrajan, A., Humphrey, M. A., Grimshaw, A. S., “Capacity and Capability Computing in Legion”, *2001 International Conference on Computational Science*, May 2001.
[16] Neulinger, N., “CGIWrap: User CGI Access”, `cgiwrap.unixtools.org`.
[17] Novotny, J., Tuecke, S., Welch, V., “Initial Experiences with an Online Certificate Repository for the Grid: MyProxy”, *High Performance Distributed Computing 10*, August 2001.
[18] Richter, J., “Custom Performance Monitoring for your Windows NT Applications”, *Microsoft Systems Journal*, August 1998.
[19] Snir, M., Otto, S., Huss-Lederman, S., Walker, D. W., Dongarra, J., *MPI: The Complete Reference*, MIT Press, 1998.
[20] Wall, L., Christiansen, T., Schwartz, R. L., *Programming Perl*, O’Reilly & Associates, ISBN: 1-56592-149-6, 1996.