Title: A Java Commodity Grid Kit

Authors: Gregor von Laszewski, Ian Foster, Jarek Gawor, Peter Lane

Address: Argonne National Laboratory, Argonne, IL 60439, U.S.A.

E-mail: gregor@mcs.anl.gov

Date: November, 2000

Version: Final v0.5

Journal: Concurrency: Experience and Practice, to be published 2001

Contents

A Java Commodity Grid Kit

GREGOR VON LASZEWSKI, IAN FOSTER, JAREK GAWOR, AND PETER LANE

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, U.S.A.

SUMMARY

Developing advanced applications for the emerging national-scale "Computational Grid" infrastructures is still a difficult task. Though Grid services are available that assist the application developers in authentication, remote access to computers, resource management, and infrastructure discovery, they provide a challenge because these services may not be compatible with the commodity distributed-computing technologies and frameworks used previously.

The Commodity Grid project is working to overcome this difficulty by creating what we call Commodity Grid Toolkits (CoG Kits) that define mappings and interfaces between Grid and particular commodity frameworks. In this paper, we explain why CoG Kits are important, describe the design and implementation of a Java CoG Kit, and use examples to illustrate how CoG Kits can enable new approaches to application development based on the integrated use of commodity and Grid technologies.

1 Introduction

The explosive growth of the Internet and of distributed computing in general has led to rapid technology development in several domains. In the world of commodity computing, a broad spectrum of distributed computing technologies (i.e., Web protocols [19], Java [17], Jini [3], CORBA [8], DCOM [23], etc.) has emerged with revolutionary effects on how we access and process information. Simultaneously, the high-performance computing community has taken big steps toward the creation of so-called *Grids* [10], advanced infrastructures designed to enable the coordinated use of distributed high-end resources for scientific problem solving.

These two worlds of what we will call "commodity" and "Grid" computing have evolved in parallel, with different goals leading to different emphases and technology solutions. For example, commodity technologies tend to focus on issues of scalability, component composition, and desktop presentation, while Grid developers emphasize end-to-end performance, advanced network services, and support for unique resources such as supercomputers. The results of this parallel evolution are multiple technology sets with some overlaps, much complementarity, and some obvious gaps.

In this context, we believe that it is timely to investigate how the worlds of commodity and Grid computing can be combined. Hence, we have established the *Commodity Grid (CoG) project*, with the twin goals of (a) enabling developers of Grid applications to exploit commodity technologies wherever possible and (b) exporting Grid technologies to commodity computing (or, equivalently, identifying modifications or extensions to commodity technologies that can render them more useful for Grid applications).

A first activity being undertaken within the CoG project is the design and development of a set of Commodity Grid Toolkits (CoG Kits)[29], which we define as follows:

Definition: A Commodity Grid Toolkit (CoG Kit) defines and implements a set of general components that map Grid functionality into a commodity environment/framework.

Hence, we can imagine a Web/CGI CoG Kit, a Java CoG Kit, a CORBA CoG Kit, a DCOM CoG Kit, and so on. In each case, the benefit of the CoG Kit is that it enables application developers to exploit advanced Grid services (resource management, security, resource discovery) while developing higher-level components in terms of the familiar and powerful application development frameworks provided by commodity technologies. In each case, we also face the challenge of developing appropriate interfaces between Grid and commodity concepts and technologies—and, if similar Grid and commodity services are provided, reconciling competing approaches.

The initial focus of our work in this area is on a Java CoG Kit. Often the question arises, Why use Java for Grid computing? Considering Java for Grid programming has several compelling advantages. We summarize here ten principal answers to this question.

1. The Language: Java as a programming language offers some features that are beneficial for large-scale software engineering projects such as packages, object-oriented approach, single inheritance, garbage collection, and unified data formats. Since threads and concurrency control mechanisms are part of the language, a possibility exists to express parallelism directly on the lowest user level in Java.

2. The Class Library: Java provides a wide variety of additional class libraries including essential functions, such as the availability to perform socket communication and access SSL, as needed for Grid computation.

3. The Components: A component architecture is provided through JavaBeans and Enterprise JavaBeans to enable component based program development.

4. The Deployment: Java's bytecode allows for easy deployment of the software through Web browsers and automatic installation facilities.

5. The Portability: Besides the unified data format Java's bytecode guarantees portability known under the term "write-once-run-anywhere."

6. The Maintenance: Java contains an integrated documentation facility. Components that are written as JavaBeans can be integrated within commercially available IDEs (Interface Development Environments).

7. The Performance: Well respected vendors have demonstrated that the performance of many Java applications can currently come close to that of C or FORTRAN.

8. The Gadgets: Java-based smart cards, PDAs, and smart devices will expand the working environment for scientists using the Grid.

9. The Industry: Scientific projects are sometimes required to evaluate the longevity of a technology before it can be used. Strong vendor support helps make Java worthy of consideration for Grid applications.

10. The Community: Universities all over the world are teaching Java to their students.

In the rest of this article, we first review briefly some Grid technologies, then use an example to illustrate what capabilities we wish the Java CoG Kit to provide, and finally present technical details on the Java CoG Kit design.

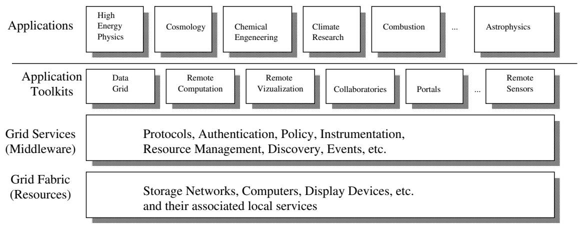


Figure 1: The integrated Grid architecture has four main categories.

2 Grids and Grid Technologies

The scientific problem-solving infrastructure of the next century will support the coordinated use of numerous distributed heterogeneous components, including advanced networks, computers, storage devices, display devices, and scientific instruments. The term "The Grid" is often used to refer to this emerging infrastructure [10]. NASA's Information Power Grid and the NCSA Alliance's National Technology Grid are two contemporary projects prototyping Grid systems; both build on a range of technologies, including many provided by the Globus project in which we are involved.

Future applications that will use Grid infrastructures will range from tomorrow's equivalent of today's "secure shell" and Web browsers to more sophisticated collaborative tele-immersive engineering, distributed petabyte data analysis, and real-time instrument control systems. These various applications will share a common need to couple devices that have not traditionally been thought of as part of the network. This need is motivating the development of a broad set of new services beyond those provided by today's Internet. These Grid services will provide the security, resource management, data access, instrumentation, policy, accounting, and other services required for applications, users, and resource providers to operate effectively in a Grid environment.

Figure 1 illustrates the structure of what we term the Integrated Grid Architecture [9], which comprises four general types of components. The *Grid Fabric* provides resource-specific implementations of basic mechanisms required for Grid operation, for example, advance reservation mechanisms in a supercomputer scheduler or storage system, or quality-of-service mechanisms in a network router.

These Grid Fabric capabilities enable the construction of resource-independent and application-independent *Grid Services*. One example is an information service, which provides uniform access to information about the structure and state of Grid resources; another example is an authentication and authorization service, which provides mechanisms for establishing identity, creating delegatable credentials, and so forth. These Grid Services are often termed "middleware": they typically involve distributed state and can be viewed as a natural evolution of the services provided by today's Internet.

Grid Fabric capabilities and Grid Services in turn enable the creation of more application-specific services and *toolkits*: for example, distributed data management capabilities to support the creation of data-intensive applications, or flow management capabilities to support the creation of collaborative work environments. These services and toolkits are then used to implement applications.

The significance of Grid infrastructures for application developers is that they greatly enhance the capabilities that can be taken for granted when developing applications. For example, a Grid-wide information service means that resource discovery and characterization become possible; hence, applications can reliably expect to discover required resources at runtime, rather than requiring resource choices to be fixed or provided by the user. Similarly, remote computation control interfaces provided in the Grid Fabric mean that, having discovered a suitable remote computer, a user can schedule, monitor, and control a computation without needing to know the idiosyncratic details of local mechanisms.

3 A Motivating Example for CoG Kits: Science Portals

We use an example to illustrate the role that we expect CoG Kit capabilities to play in future Grid/commodity architectures and the technology developments required to realize this promise. The example is an instantiation of what some call a "science portal": an access point (e.g., desktop, browser, palm device) designed to facilitate scientific research in a particular discipline by providing seamless access to a wide range of information and computational resources.

3.1 Science Portal Scenario

We consider a "Midwest Climate Change Portal" that provides access to computational and data resources relating to regional impacts of global climate change. Such a portal serves a variety of users with different needs and interests, for example, climate researchers, weather forecasters, students, traffic control agencies and services, and farmers. We consider two usage scenarios.

Researcher: A researcher interested in impacts of climate change on cranberry bog yields in Wisconsin uses the portal to discover relevant datasets and models. He quickly puts together a description of his required data, using a graphical editor. This description is transformed automatically in a sequence of computations and lookups in order to obtain the desired data. Existing software infrastructure must be seamlessly integrated into the set of tools used by the researcher to derive results. The results of such an interaction can be viewed using browsers while preparing and invoking further analysis on the data. The results are discussed and interpreted with the help of colleagues during interactive sessions and then are posted to an electronic notebook and are prepared for the use of other interested parties.

Farmer: A farmer uses the portal when planning which crop he should grow on his fields. His questions focus on whether, when, and how to use his land in order to achieve a maximum benefit over years. Naturally, he needs to obtain a seasonal forecast allowing him to determine the best time for planting the crop. Electronic microsensors distributed in his ground help to steer the use of fertilizers during the growth period. Sensor data is fed into a database accessible by scientists, allowing for feedback to check for model accuracy. Access points to the portal include computer terminals in electronically enhanced farm buildings and also specialized input and output devices that allow for the installation in, for example, a lightweight wireless device to access a useful subset of the information in the field. The farmer's portal also provides access to other services and information sources, for example, financial market monitoring services that observe the fluctuation of the value of the crops and give advice that may result in greater profits (Figure 2).

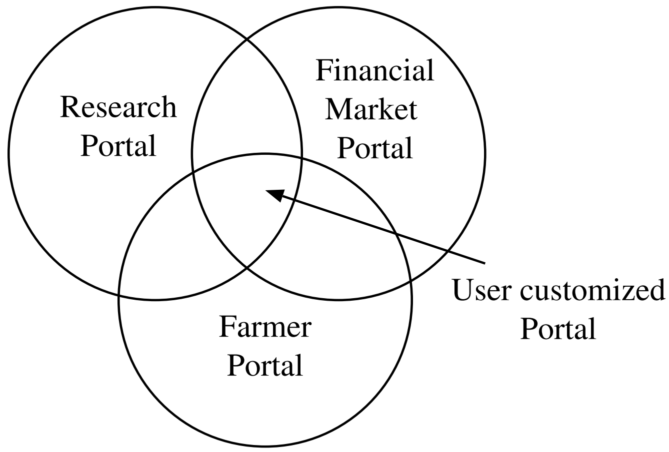


Figure 2: Multiple portals provide access to overlapping functionality, with a particular portal specialized to the requirements of its user.

3.2 Science Portal Requirements

The creation of science portals such as those just described requires the integration of many technologies from different fields. We will typically provide access to a wide variety of data; hence, we must be able to *access and communicate with a wide range of information sources*. The complex calculations performed on this data require the ability to access compute resources with significant computational capabilities. We may also require access to proprietary software loaded on remote machines. Thus, the *ability to incorporate remote computational resources* is required. Interactive use can require that computational and data resources be accessed via high-performance networks. Thus, it is beneficial to be able to enforce *performance guarantees* for data transfers and computations in order to support increased performance demands during interactive sessions.

The success of a science portal is also measured by its usability and acceptance in the community. Hence, we require environments that allow rapid prototyping of both complete applications and new components that can be shared with other users. The *ability to rapidly create portable user interfaces* is particularly critical. These requirements overlap strongly with two types of technology:

- ▷ *Commodity technologies* that emphasize ease of use and code reuse in local (especially desktop) environments: GUI components, component libraries, scripting languages, industry-accepted distributed computing frameworks, industrial-strength database servers, object-oriented programming languages and frameworks, and the like.

- ▷ *Grid technologies* that emphasize effective operation in large-scale, multi-institutional, wide area environments: access to remote computation, information services, high-speed data transfers, special protocols (e.g., multicast), and gateways to local authentication schemes.

These considerations lead to the question that has motivated the research reported in this paper: How can commodity and Grid technologies interface and integrate so as to adhere to interoperability — and, ideally, to enhance the capabilities of both? For example, we might decide to use CORBA for application development but also want to use Grid services for scheduling and managing computations on a supercomputer. Or, if we are using Java, then Jini might appear to be a good mechanism for resource discovery: but then we face the problem of accessing data stored in the extensive (currently LDAP-based) Grid information service. The interactions can be complex and require significant thought and effort to include them into a Grid-based information service. Yet the technology base that exists in each case is sufficiently large and robust that exploiting these existing mechanisms leads to a significant enhancement of both Grid and commodity-based technologies.

4 Commodity Grid Toolkits

The combination of commodity and Grid technologies can, in principle, enable exciting new applications that tie advanced network-accessible resources into the commodity desktop. Our goal in the Commodity Grid project is to enable these opportunities to be realized in practice. Our research approach involves an iterative process of definition, development, and application of Commodity Grid Toolkits (CoG Kits): sets of general components that map Grid functionality into specific commodity environments or frameworks. The word *map* is important: the integration of Grid and commodity technologies is not simply an interface definition problem but rather is concerned with how Grid concepts and services are best expressed in terms of the concepts and services of a particular commodity framework. To take a simple example, in the Globus Grid toolkit on which we are building our prototypes, remote computation management is handled via a procedural API and callbacks; in the Java CoG Kit, the same functionality is provided via a Job object and Java events.

The requirements of the science portals and other applications have motivated us to explore mappings to several languages. In particular, we are exploring *Perl* and *Python*, in order to support easy prototyping and Web-based programming based on CGI scripts and *Java*, in order to support graphical user interface development, ease of programming, and the ability to run many Grid services through Java-enabled Web browsers.

We also need to address the issue of accessing Grid services through high-level distributed computing frameworks defined by industry, so as to allow integration of common off-the-shelf tools and development environments. Hence, we consider the Common Object Request Broker Architecture (CORBA), and the Distributed Component Object Model (DCOM).

We must also consider the environments and platforms for which the application is targeted. This might include *low cost devices* such as PDAs and PDA-enhanced communication devices, to provide cheap input devices and monitoring tools, as well as specialized visualization environments such as CAVE. Support for operating systems such as MSWindows is essential to address the large market surrounding Microsoft infrastructure, as is support for UNIX/Linux to address the server side and the large amount of free and commercial-grade scientific code available for workstations and supercomputers.

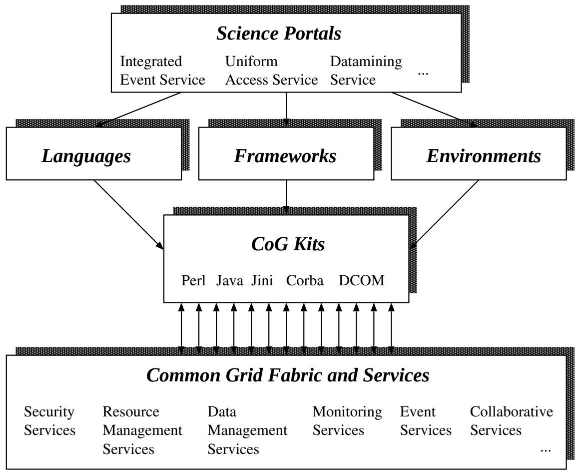


Figure 3: CoG Kits provide a mapping between computer languages, frameworks, and environments, and Grid Services/Fabric. Together, services, languages, frameworks, and environments build a powerful development tool for building Grid-enhanced applications.

5 Java CoG Kit

In the rest of this paper we focus our attention on our Java CoG Kit prototype and explain how it enables us to access Grid services provided by the Globus toolkit. Because of the large number of packages and classes required to expose the necessary functionality of the Globus toolkit, we focus in this paper on a subset of all available classes that we deem most useful for the development of Java-based Grid applications. We intend to facilitate the development of future components as a community project with the Java CoG Kit. To support an iterative process of definition, development, and application of a Java CoG Kit in collaboration with other teams, we classify components as depicted in Figure 4. This categorization provides the necessary subdivision in order to coordinate such a challenging open community software engineering task. This categorization is based on an increased functionality of the components. Each subsequent category reuses the lower-level components.

Low-Level Grid Interface Components provide mappings to commonly used Grid services. This includes for example, the access to services using the the *Grid Security Infrastructure* (GSI), which delivers a secure method of accessing remote resources[6]. Other examples are the *Grid information service* (the Globus Metacomputing Directory Service, MDS) [27], which provides Lightweight Directory Access Protocol (LDAP) [18] access to information about the structure and state of Grid resources and services; *resource management services*, which support the allocation and management of computational and other resources (via the Globus GRAM services [15]); and *data access services* (for example, via the Globus GASS service [4] and GSI enabled FTP). No graphical components are available in these low-level components.

Low-Level Utility Components are utility functions designed to increase the functionality beyond those provided by the C implementation of the Globus toolkit. This

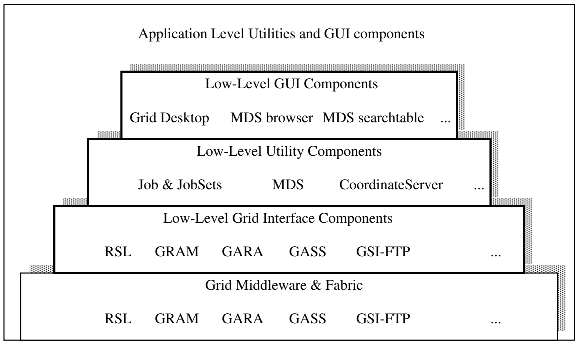


Figure 4: Applications and more complex components can be built with the help of the CoG Kit. Components are classified here based on their role.

functionality is exposed through services and classes that can be reused by many users. Examples are components that use the MDS to find all compute resources that a user can submit to that prepare and validate a job specification while using the extended markup language (XML) [14] or the Globus job submission language (RSL) that locate the geographical coordinates of a compute resource or that test whether a machine is alive. No graphical components are available in these low-level utility components.

Common Low-Level GUI Components provide a set of low-level GUI components that can be reused by application developers. Examples for such components are LDAP Attribute Editors, RSL editors, LDAP browsers, and search components.

Application-specific GUI Components simplify the bridge between applications and the basic CoG Kit components. Examples are a stockmarket monitor, a graphical climate data display component, or a specialized search engine for climate data.

For each of the above classes we will provide in this paper exemplary Java CoG Kit components and code fragments.

6 Java CoG Kit Implementation

Figure 5 shows how our Java CoG Kit is used in practice. This Java program skeleton, which forms part of a climate portal demonstrates how simple it is to build portal-specific services when accessing a variety of basic Grid services through the Java CoG Kit. In this example, an appropriate machine is selected for execution, data for an instantiation of the climate model is located and downloaded to the machine, and the climate model is executed on that machine. The program generates an output file in GrADS [13] format, a well-known format for storing three-dimensional climate-related data. Throughout the remainder of the paper we will expand this example as we introduce various Java CoG Kit components.

```
// Step 0. Initialization
   String mdsServer = "mds.globus.org";
     MDS mds=new MDS(mdsServer,"389");
// JOB SUBMISSION
//Step 1. Search for an available machine
     result = mds.search ("o=Grid",
        "&((object-
class=GridComputeResource)(freenodes=64))",
        "contact");
// Step 1.a) Select a machine
     machineContact = <select the machine with minimal execution time from
                         the contacts that are returned in result>
// Step 2. Prepare the data for the experiment
// Step 2.a) Search for the climate data and return
//            the  attributes: server,port,directory,file
     dn = mds.search
         ("(objectclass=ClimateData)(year=2000)
           (region=midwest)",
          "dn", MDS.SubtreeScope);
   result = mds.lookup (dn, "server port directory file");
// Step 2.b) download the data to the machine
   filename = result.get("filename");
     sourceURL = result.get("server")+":"
              + result.get("port")+"/"
              + result.get("directory")+"/"
              + filename;
   destinationURL = "gsi-ftp://"
              + "machineContact"      // destination
              + "~/"                  // directory
              + filename;              // filename
   UrlCopy.copy (sourceURL, destinationURL);
//   Step 3. Prepare a description for running the model
    RSL rsl = new RSL("(executable=climateModel)(processors=64)"
                + "(arguments=-grads)(arguments=-
out map.grads)"
                + "(arguments=-in " + filename +")");
// Step 4. Submit the program
    GramJob job = new GramJob(rsl.toString());
   job.addJobListener(new GramJobListener() {
      public void stateChanged(GramJob job) {
         // react to job state changes
      }
   });
   try{
     job.request(machineContact);
   } catch (GramException e) {
      // problem submitting the job
  }
```

Figure 5: This sample script demonstrates how we access basic Grid services with the help of the Java CoG Kit. Here data for a climate model is located, an appropriate machine is selected, and the climate model is executed on that machine.

6.1 Low-Level Grid Mappings

In this section we enumerate a subset of packages that provide the interface to the low-level Grid services and application interfaces. These packages are used by many users to develop Java-based programs in the Grid. We will describe only the general functionality of these packages, as it is beyond the scope of this paper to explain every class and method. For a complete list of the classes and methods we refer to the distribution [30].

RSL. The package *org.globus.rsl* provides methods for creating, manipulating, and checking the validity of the RSL expressions used in Globus [15] to express resource requirements. As shown in Step 3 of Figure 5, the arguments to a *new* call include parameters that specify both characteristics of the required resources and properties of the computation.

GRAM. The package *org.globus.gram* provides a mapping to the Globus GRAM services [15], which allow users to schedule and manage remote computations. Specifically, various classes and methods allow users to submit jobs, bind to already submitted jobs, and cancel jobs on remote computers. Other methods allow users to determine whether they can submit jobs to a specific resource (through a Globus gatekeeper) and to monitor the job status (*pending, active, failed, done,* and *suspended*).

As shown in Step 4 of Figure 5 the class Gram is used to create a job with an RSL string describing the job and a machine contact that determines on which machine the job is requested for execution. Our Java mapping differs from that provided in Globus for C through the introduction of a formal job object.

Moreover, our implementation utilizes the sophisticated event model in Java and transfers the C callbacks into equivalent Java events. In Java one can now use threads in order to "listen" to a particular event that can trigger further actions. A Java interface `GramJob-Listener` that contains the method `stateChanged(GramJob job)` can be used to define customized job listeners that can be added with the GramJob method `addListener(GramJobListener listener)`.

Previous efforts to integrate Globus job management into Java have mostly been based on submission of commands through the runtime method in Java. The result has been less portable code. Moreover, although such an approach provides an easy way of abstraction, it significantly restricts the capabilities of the Globus middleware because it does not allow one to utilize the enhanced features of Globus GRAM (see, e.g. projects such as Gateway [16], and Webflow [1]). Furthermore, portals require that client software be written in pure Java, without the execution of native programs, so as not to violate the Java security model (see, e.g., projects such as CCAT [5] and Cactus [2]).

MDS. The package *org.globus.mds* simplifies the access to the MDS [27], which is an important part of the Globus information service. Its functions include (a) establishing a connection to an MDS server, (b) querying MDS contents, (c) printing, and (d) disconnecting from the MDS server. The package provides an intermediate application layer that can be easily adapted to different LDAP [18] client libraries, including JNDI [21], Netscape SDK [20], and Microsoft SDK [24].

As shown in Step 0 of Figure 5, the parameters to initialize the MDS class are the DNS name of the MDS server and the port number for the connection A search is performed in Step 2.a; the first parameter specifies the top level of the tree in which the search is performed, the second parameter specifies the LDAP query, and the third parameter specifies the scope, that is, for how many levels in the tree the search should continue (in our case only the next level). Search results can also be stored in a NamingEnumeration provided by JNDI.

Another MDS service transforms the information into an XML based format called MDSML. This format is currently developed as part of the Grid Forum [31] and provides a convenient way to exchange data between different services.

GASS. The Global Access to Secondary Storage (GASS) service [4] simplifies the porting and running of applications that use file I/O, eliminating the need to manually log onto sites and ftp files or to install a distributed file system. The package *org.globus.gass* provides an essential subset of GASS services to support the copying of files between computers on which the Grid Services are installed. The method *copy(String fromUrl, String toUrl)* copies a file from one file server to another. Hence, it can be used to *stage* data to a remote machine prior to job execution (Figure 5).

GSI. The *Grid Security Infrastructure* (GSI), delivers a secure method of accessing remote resources [6]. It enables a secure, single sign-on capability, while preserving site control over access control policies and local security infrastructure. The package *org.globus.security* contains a set of classes for developing client and server side applications requiring access to GSI enabled resources.

GSI-FTP. The GSI-FTP service provides a secure way to transfer files in a Grid environment. It extends the standard FTP protocol and supports third-party control of data transfer. Thus, files can be moved directly between servers while the user controls the transfer from a third machine. The package *org.globus.io.ftp* provides all necessary functions such as putting and getting files and listing directories.

GARA. The Globus Architecture for Reservation and Allocation (GARA) provides advanced reservations and end-to-end management for quality of service on different types of resources, including networks, CPUs, and disks [11]. The package *org.globus.gara* implements an elementary client side interface to GARA. Thus it enables development of services that need quality of service.

6.2 Low-Level Utilities

The low-level utility classes currently defined in the CoG Kit provide an abstract datatype representing acyclic graphs and basic XML parsing routines. The graph class is used, for example, to access dependencies between jobs, a major requirement for science portal applications. The XML classes are used to provide transformations between different data formats. Using XML has the advantage that a Document Type Definition (DTD) that is defined for these data formats can be used to verify whether a record to be transmitted is well formed before it is sent to a server. Thus the load on servers can be dramatically reduced. The availability of a dependency between jobs is a significant extension to the existing Globus low-level application interface. In addition, we have defined a general concept of a *machine* and *job broker* interface. This enables a programmer to define a customized selection of machines and jobs dependent on his demand. We have used this technology as part of a high-throughput broker that is implemented in Java but can also exposed through

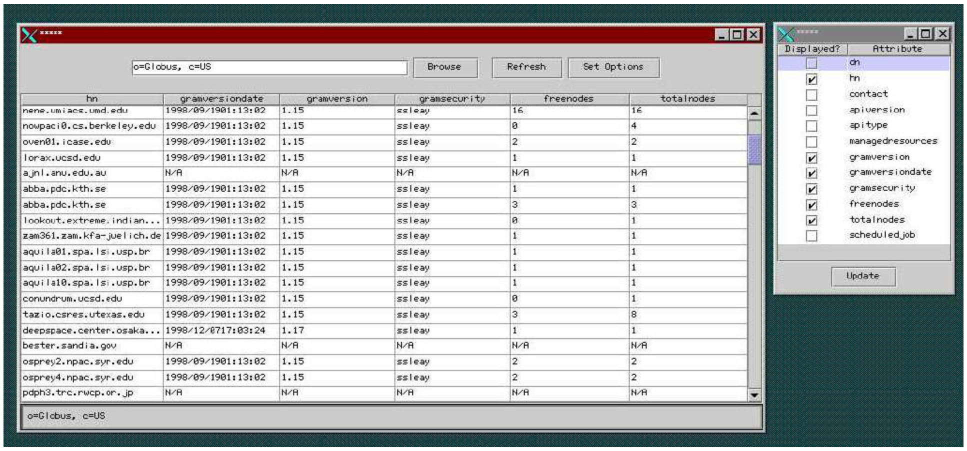


Figure 6: The MDS search table can be used to display selected MDS information in a tabular form. The search string can be specified, and attributes can be selected easily to customize the table.

```
// Step 0: Initialize the table
    MDSsearchTable table = new MDSsearchTable (mds);
// Step 1: perform a search in the MDS to request data to be displayed
    table.search ("(objectclass=GridComputeResources)",
            "hn gramversion contact");
// Step 2: display and update the table
     table.show();
// Step 3: return the selection
    String machineContact = table.getSelection("contact");
```

Figure 7: The program shows the ease of use of the Graphical User Interface for selecting a Grid contact string (compare Figure 6).

CORBA objects. The GECCO application introduced in Section 6.4 utilizes the Java-based machine and job brokers.

6.3 Low-Level GUI Components

The Java CoG Kit low-level GUI components provide basic graphical components that can be used to build more advanced GUI-based applications. These components include text panels that format RSL strings, tables that display results of MDS search queries (Figures 6 and 7), trees that display the directory information tree of the MDS, and tables to display network performance data. Each component can be customized and is available as a JavaBean. In future releases of the Java CoG Kit it will be possible to integrate the bean in a Java-based GUI composition tool such as JBuilder or VisualCafe.

6.4 High-Level Graphical Application

High-level graphical applications combine a variety of CoG Kit components to deliver a single application or applet. Naturally, these applications can be combined in order to provide even greater functionality. The user should select the tools that seem appropriate for the task. As an example we will describe one of the many possible high-level components.

GECCO. The Graph Enabled Console COmponent (GECCO) is a graphical tool for specifying and monitoring the execution of sets of tasks with dependencies between them [28][26]. Specifically it allows one to

1. specify the jobs and their dependencies graphically or with the help of an XML-based configuration file,
2. debug the specification in order to find erroneous specification strings before the job is submitted, and
3. execute and monitor the job graphically and with the help of a log file.

As shown in Figure 8, each job is represented as a node in the graph. A job is executed as soon as its predecessors are reported as having successfully completed. The state of a job is animated with colors. It is possible to modify the specification of the job while clicking on the node: a specification window pops up allowing the user to edit the RSL, the label, and other parameters. Editing can also be performed during runtime (job execution), hence providing for simple computational steering.

Grid Desktop. The Grid Desktop component is intended to showcase the powerful graphics functions a user can utilize while using the Java CoG Kit. It is a graphical tool that mimics the familiar desktop environments of Windows. Here users can place icons that represent compute resources and job specifications. Dragging a job icon onto a machine icon will start the job on that machine by executing the corresponding RSL string. Thus, job submission on the Grid Desktop is as easy as generating new job icons and dragging them onto the desired machine icon. Furthermore, we have augmented the machine icons with a history function that lets users list information about previously started jobs. Clicking on the standard output field of this machine history table will fetch the output file automatically to the local client and display it in a separate window. Figure 9 shows the execution of a job listing the home directory on a remote machine called pitcairn. This component is able to interact with some other advanced components such as a component that is used to browse the contents of the MDS. Dragging a Globus resource manager listing from the LDAP browser to the Grid Desktop will automatically create a new icon for this new resource. Using the drag-and-drop feature of Java will allow us to integrate many graphical components independently developed by grid component developers. The exchange between components is done while providing a commonly defined interface format that is preferably formulated in XML.

7 Installation and Upgrading

An important function that must be available as part of the Java CoG Kit is the ability to install and upgrade the software that accesses the various services. Using Java will provide

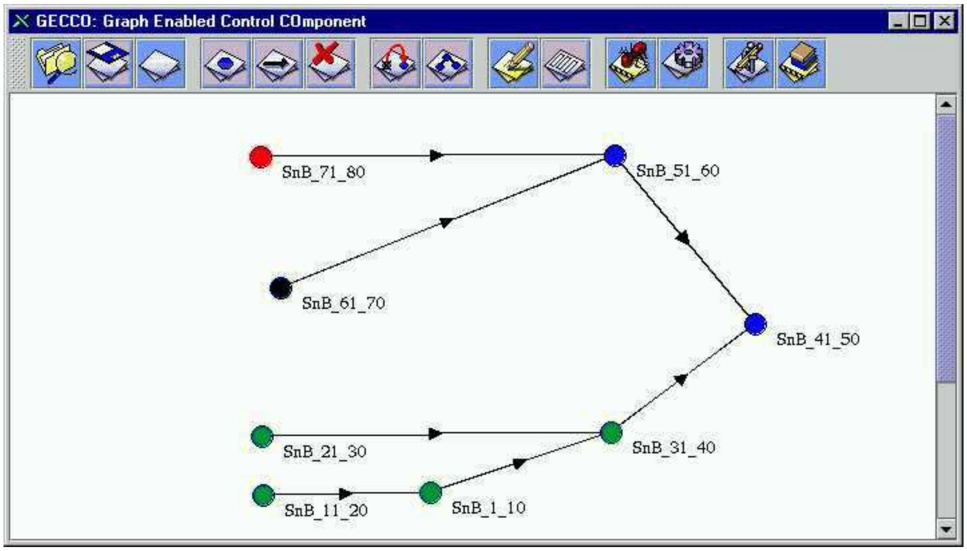


Figure 8: The Grid Enabled Console COmponent (GECCO) allows the user to specify dependencies between tasks that are to be executed in the Grid environment. Here we show a graph created for a crystallography application *Shake 'n Bake*.

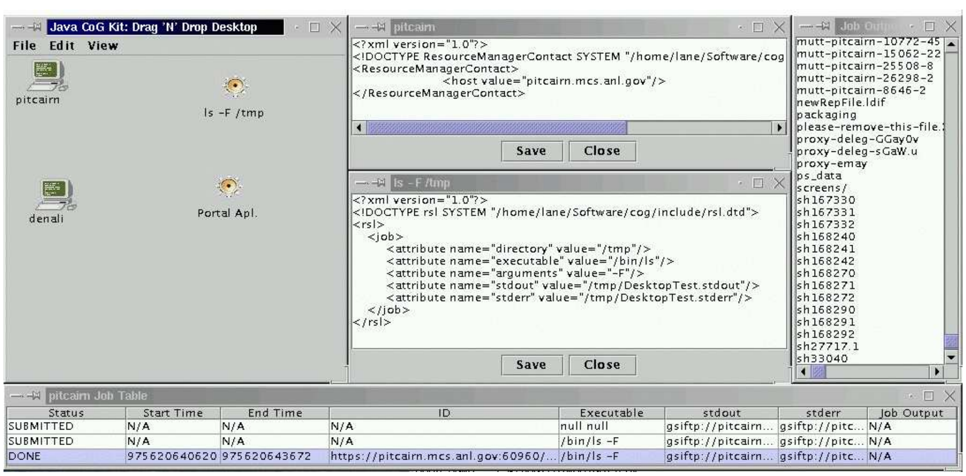


Figure 9: The Grid Desktop allows users to specify their own view of the Grid while placing icon representations of remote machines and jobs on a desktop. Dragging job icons onto machine icons will start the execution.

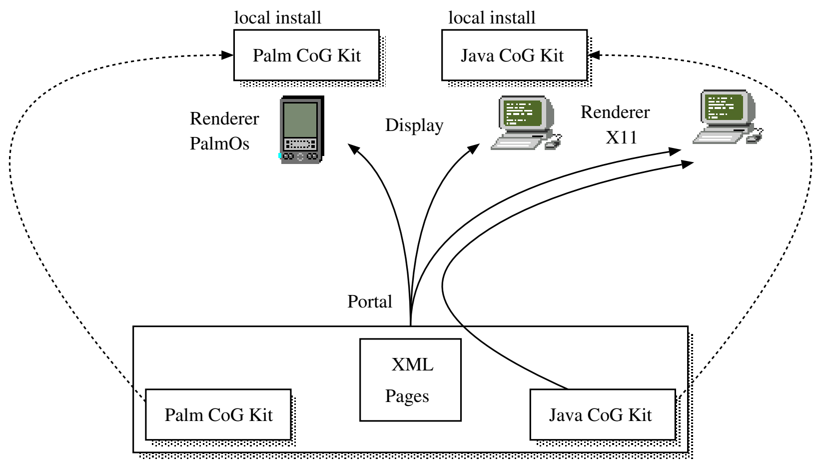


Figure 10: The installation of the CoG Kit onto a client can be done prior to the start of the application as a stand-alone application or the installation of a library or during an on-demand execution.

us with several options for deploying client software. In addition to traditional methods of delivering client software to be installed and configured prior to its use, we can develop thin-client software, which can be dynamically installed or updated as well as loaded at time of use.

Preinstallation of the software in the form of a stand-alone application or a library is convenient for applications that would take too long to be installed via a network connection (Figure 10). This strategy is today used by many commercial portals as part of their access software, enabled with the help of browser plug-ins. Nevertheless, we recognize the fact that it is sometimes not possible to install any software on the client computer because the user does not have sufficient access to it. This requires, at the cost of additional download time, downloading the appropriate *jar* files from a well-defined URL. In both cases it will be possible to augment the jar files with authentication measures in the form of certificates. These will allow clients to identify the source of the code upon downloading our software and to verify that it can be trusted for use on their systems.

8 Future Applications

The availability of the Java CoG Kit has several advantages for developing future Grid-based applications. The assumed platform independence of Java and its increased popularity provide the basis of a promising platform in the near future. Furthermore, since Java is well established on the Windows operating system, it seems an obvious candidate for delivering a Globus server-side implementation, hence allowing jobs to be submitted to any NT machine as long as it is integrated in the Grid. More straightforward is the development of a Globus thin-client, which consists only of the necessary security routines and the communication routines to communicate with a Globus server. All previous releases of CoG components used a pull model to inquire about the state of a submitted job. Since we have changed the model to use listeners, it is now easier to write threaded Grid-based Java applications based on a push model. Projects that will benefit from this approach are, for example, Gateway [16], Webflow [1], CCAT[5], and Cactus[2].

Since the latest Java CoG components can be integrated with Java applets or applications running directly from the Web browser, it is possible to develop thin clients accessing the Grid. Projects such as WebSubmit [22] and Hotpage [25] will profit from this change. Making some components available as JavaBeans and integrating them into common off-the-shelf Java GUI building tools will provide a Grid development environment that allows Grid programming with ease. As a result of the availability of the Java CoG Kit, recent efforts to standardize the Globus delegation model in cooperation with the development of the Java CoG Kit will allow a much easier integration with existing commodity technologies.

9 Summary

Commodity distributed-computing technologies enable the rapid construction of sophisticated client-server applications. Grid technologies provide advanced network services for large-scale, wide area, multi-institutional environments and for applications that require the coordinated use of multiple resources. In the Commodity Grid project, we seek to bridge these two worlds so as to enable advanced applications that can benefit from both Grid services and sophisticated commodity development environments.

The Java Commodity Grid Toolkit (CoG Kit) described in this paper represents a first attempt at creating of such a bridge. Building on experience gained over the past three years with the use of Java in Grid environments, we have defined a rich set of classes that provide the Java programmer with access to basic Grid services, enhanced services suitable for the definition of desktop problem solving environments, and a range of GUI elements. Initial experiences with these components have been positive. It has proved possible to recast major Grid services in Java terms without compromising on functionality. Some substantial Java CoG Kit applications have been developed, and reactions from users have been positive.

Our future work will involve the integration of more advanced services into the Java CoG Kit and the creation of other CoG Kits, with CORBA, DCOM, and Python being early priorities. We also hope to gain a better understanding of where changes to commodity or Grid technologies can facilitate interoperability and of where commodity technologies can be exploited in Grid environments.

Availability

The Java Cog Kit is available in beta release from the CoG Kit Web pages [30]. The release of the components is done gradually to assure the necessary quality control of the delivered packages, classes, and methods. At present, the main distribution contains the low-level components. Besides the components described in this paper, we have an implementation of network based quality-of-service methods. For more release notes, we refer to the Web page http://www.globus.org/cog.

Acknowledgments

Many technologies and research projects are related to and important for the development of the CoG Kits. Some of them can be found in [7]. We are grateful to members of the NCSA Alliance for enlightening discussions on these topics; in particular, we thank Warren Smith, Jay Alameda, Dennis Gannon, Geoffrey C. Fox [12], and Mary Pietrowicz. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; by the National Science Foundation; and by the NASA Information Power Grid program.

References

[1] E. Akarsu, G. C. Fox, W. Furmanski, and T. Haupt. WebFlow - High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing. In *Supercomputing'98*, Orlando, FL, Nov. 1998.

[2] G. Allen, W. Benger, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, and E. Seidel. The Cactus Code: A Problem Solving Environment for the Grid. In *Proc. 9th IEEE International Symposium on High Performance Distributed Computing*, pages 253–260, Pittsburg, Aug. 2000.

[3] K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification*. The Java Technology Series. Addison-Wesley, June 1999.

[4] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A data movement and access service for wide area computing systems. In *In Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pages 78–88, Atlanta, GA, May 1999. ACM Press.

[5] R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, and M. Yechuri. A Component Based Services Architecture for Building Distributed Applications. In *Proc. 9th IEEE International Symposium on High Performance Distributed Computing*, Pittsburgh, PA, Aug. 2000.

[6] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. Design and Deployment of a National-Scale Authentication Infrastructure. *IEEE Computer*, 33(12):60–66, 2000.

[7] Computing Portals: Project Catalog. http://www.computingportals.org/projects, 1999.

[8] CORBA 2.0/IIOP Specification. http://www.omg.org/corba/c2indx.htm.

[9] I. Foster. Building the Grid: An Integrated Services and Toolkit Architecture for Next Generation Networked Applications. http://www.gridforum.org/building_the_grid.htm, July 1999.

[10] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan-Kaufmann, 1999.

[11] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *Proceedings of the 8th International Workshop on Quality of Service (IWQOS)*, pages 181–188, Pittsburgh, PA, June 2000.

[12] G. C. Fox and W. Furmanski. HPcc as High Performance Commodity Computing. http://www.npac.syr.edu/users/gcf/HPcc/HPcc.html, Dec. 1997.

[13] GrADS: Grid Analysis and Display System. http://grads.iges.org/grads/.

[14] I. S. Graham and L. Quin. *XML Specificaton Guide*. Wiley, 1999.

[15] Globus Resource Management. http://www.globus.org/toolkit/resource-management.html.

[16] T. Haupt, E. Akarsu, G. Fox, and C.-H. Youn. The Gateway System: Uniform Web Based Access to Remote Resources. *Concurrency: Practice and Experience*, 12(8):629–642, Aug. 2000.

[17] C. S. Horstmann and G. Cornell. *Core Java 2*, volume 1 and 2. Prentice Hall, 4th edition, Dec. 1999.

[18] T. Howes and M. Smith. *LDAP : Programming Directory-Enabled Applications With Lightweight Directory Access Protocol*. Technology Series. Macmillan Technical Publishing, 1997.

[19] HTTP Hypertext Transfer Protocol. http://www.w3.org/Protocols/.

[20] Netscape Directory and LDAP Developer Central. http://developer.netscape.com/tech/directory/index.html.

[21] R. Lee and S. Seligman. *JNDI API Tutorial and Reference: Building Directory-Enabled Java Applications*. The Java Series. Addison-Wesley, May 2000.

[22] R. McCormack, J. Koontz, and J. Devaney. ESeamless Computing with WebSubmit. *Concurrency, Practice, and Experience*, 11(15):949–963, 1999.

[23] D. Rogerson. *Inside COM - Microsoft's Component Object Model*. Microsoft Press, 1997.

[24] R. Schwartz. *Windows 2000 Active Directory Survival Guide: Planning and Implementation*. John Wiley and Sons, 1999.

[25] M. Thomas. The Hotpage Web Page. http://hotpage.npaci.edu.

[26] G. von Laszewski. A Loosely Coupled Metacomputer: Cooperating Job Submissions across Multiple Supercomputing Sites. *Concurency, Experience, and Practice*, 11(15):933–948, 1999.

[27] G. von Laszewski, S. Fitzgerald, I. Foster, C. Kesselman, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proc. 6th IEEE Symp. on High-Performance Distributed Computing*, pages 365–375, 1997.

[28] G. von Laszewski and I. Foster. Grid Infrastructure to Support Science Portals for Large Scale Instruments. In *Proc. Workshop on Distributed Computing on the Web (DCW)*, pages 1–16. University of Rostock, Germany, June 1999.

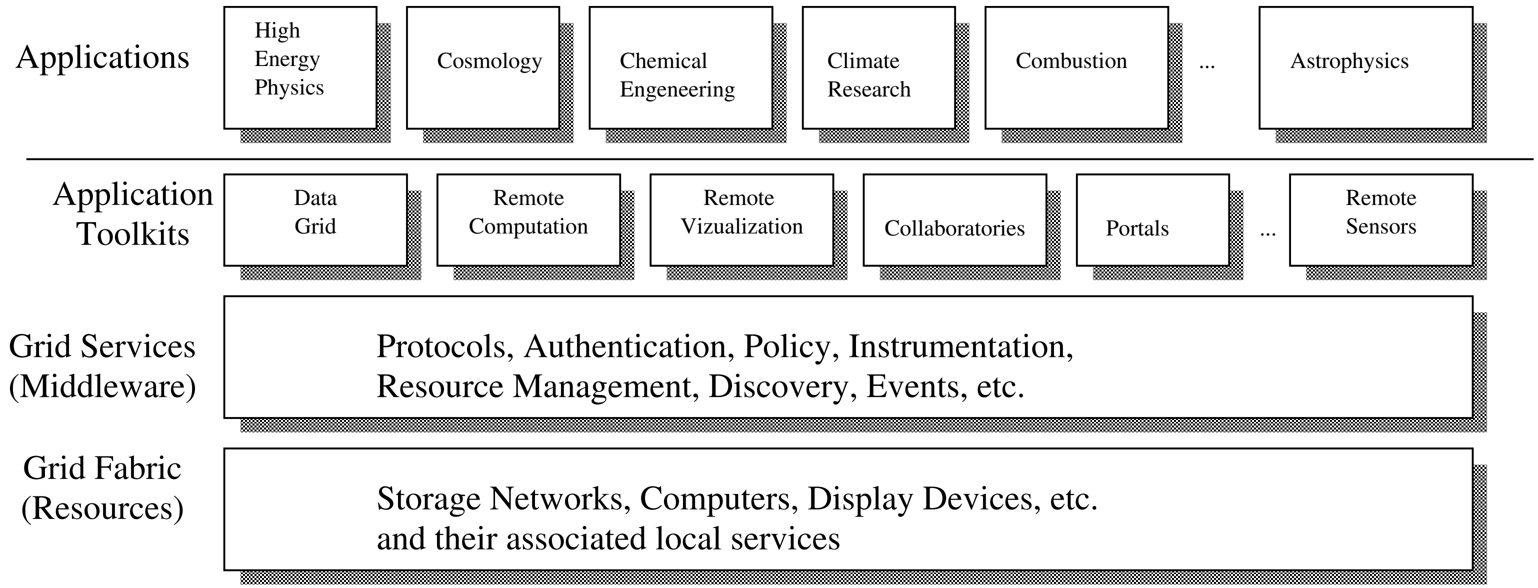
[29] G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *ACM 2000 Java Grande Conference*, pages 97–106, San Francisco, CA, June 2000.

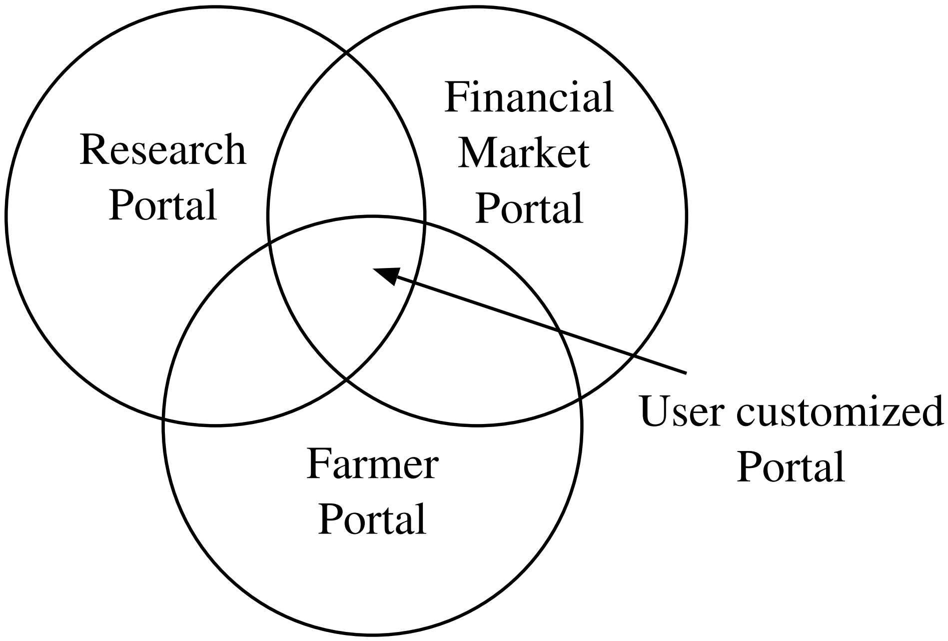
[30] G. von Laszewski, J. Gawor, and P. Lane. Java CoG Distribution. http://www.globus.org/cog, Dec. 2000.

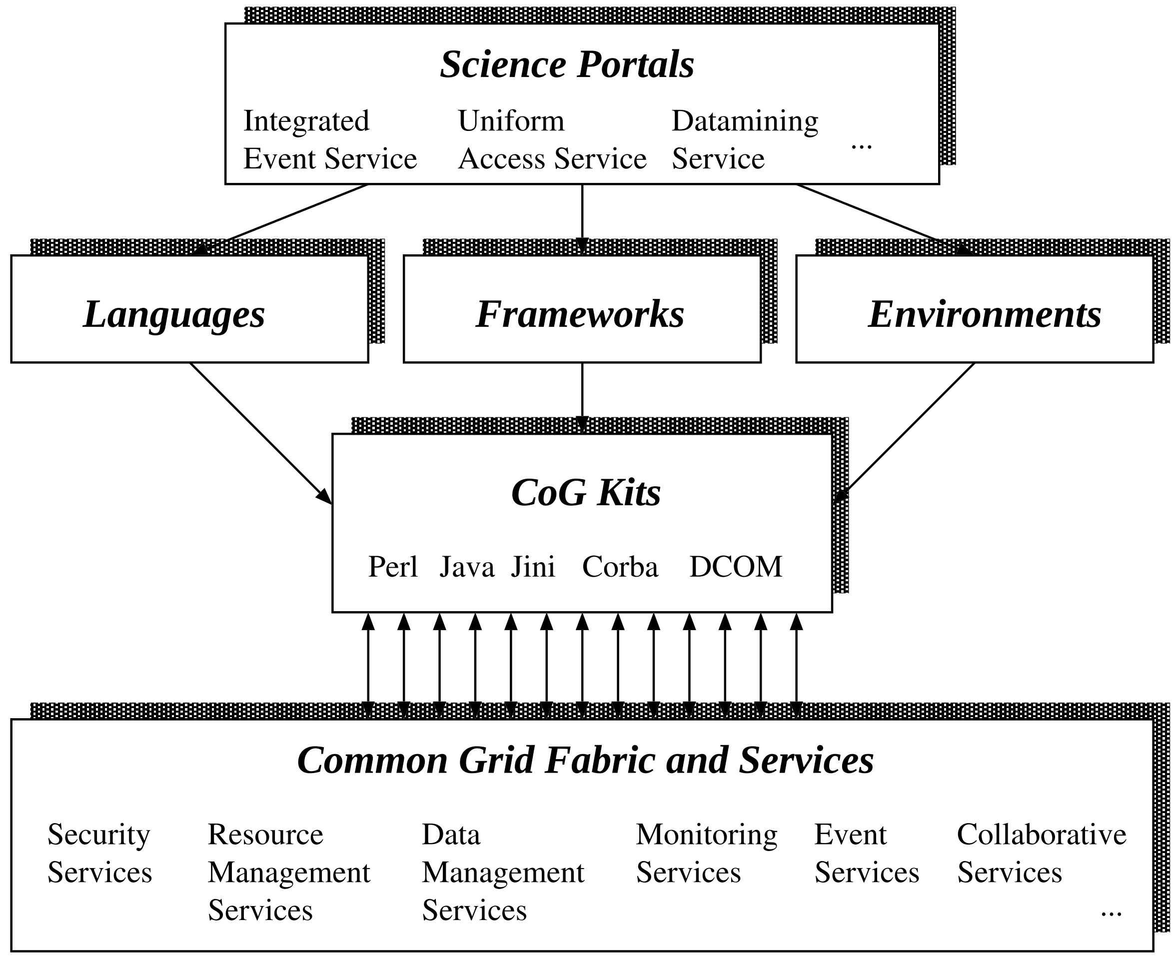
[31] G. von Laszewski and P. Lane. MDSML: An XML Binding for the Grid Object Specification. Gridforum Working Group Document GIS-WG 2, Argonne National Laboratory and Pacific Northwest Laboratory, June 2000. http://www.gridforum.org.

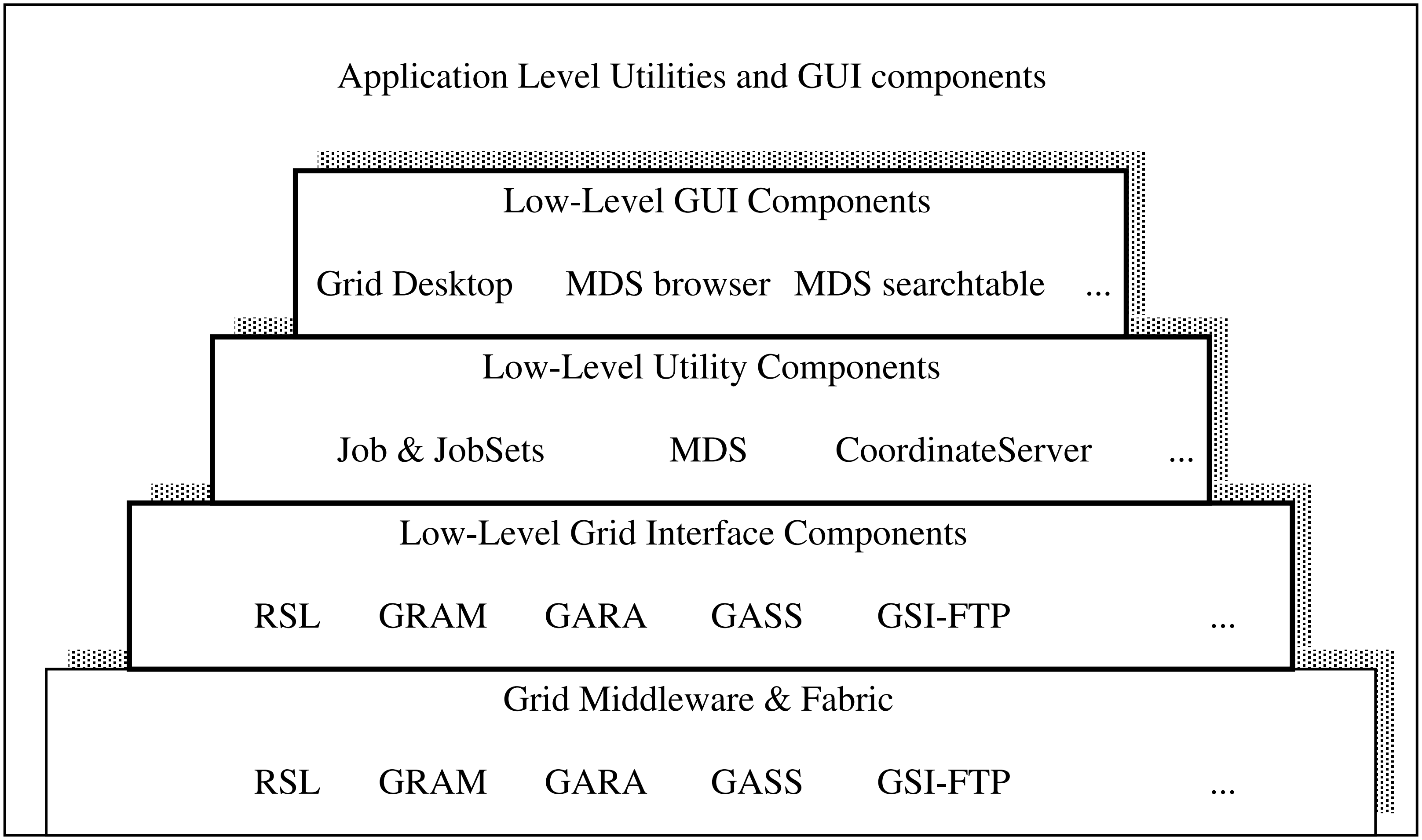
A Figures

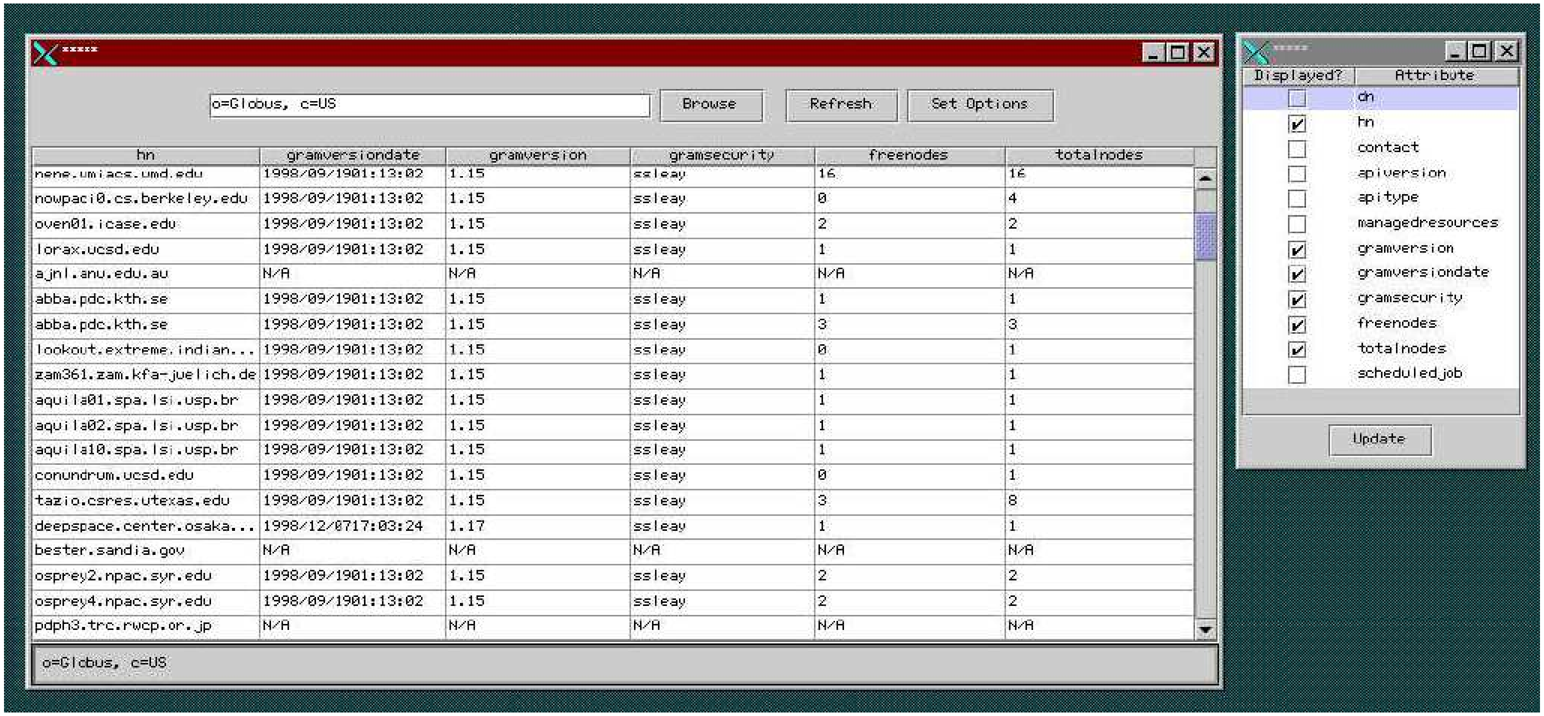
For convenience of the reproduction, some figures are included in this appendix in a larger format.

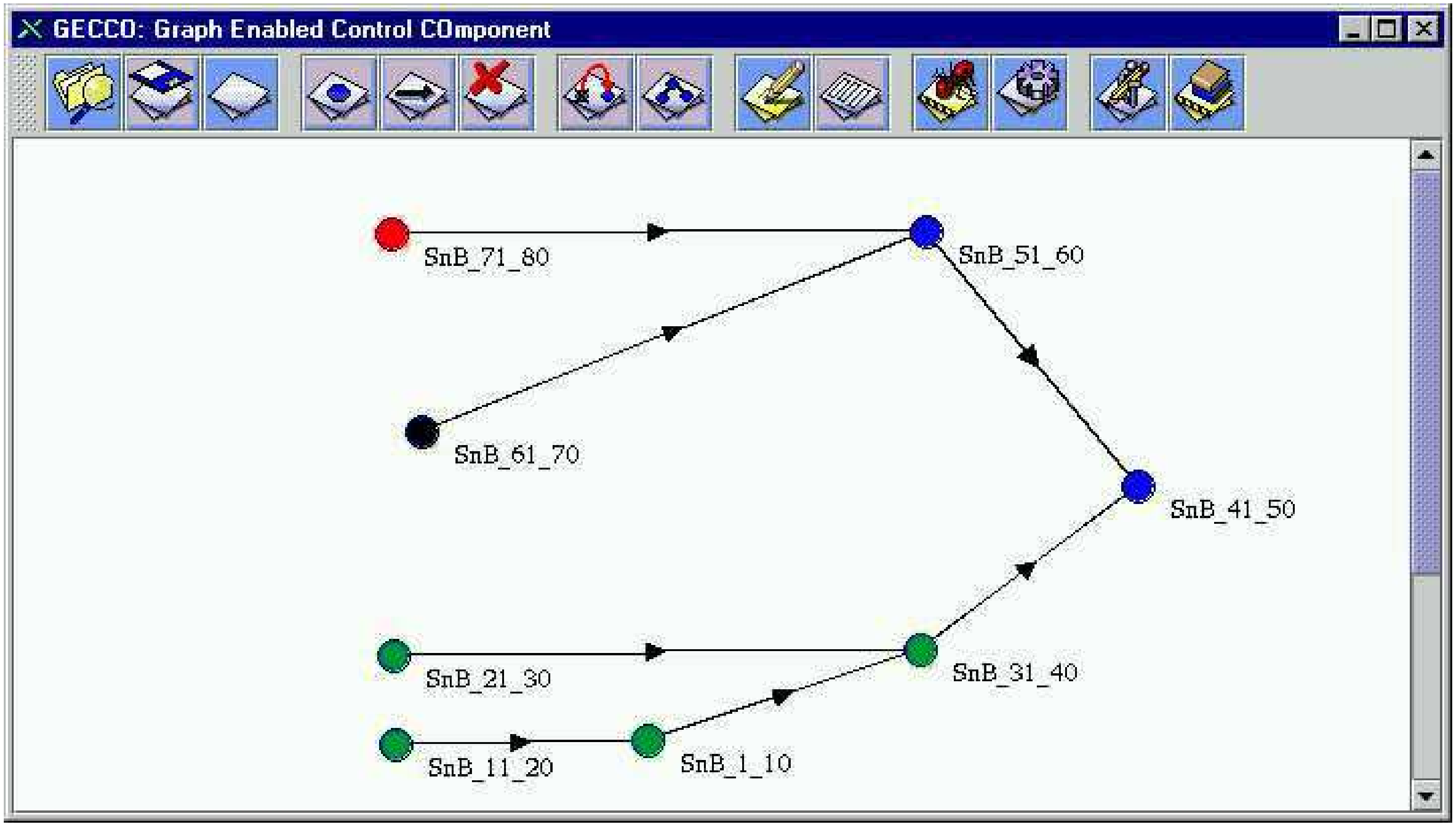

Applications
High Energy Physics
Cosmology
Chemical Engeneering
Climate Research
Combustion
...
Astrophysics
Application Toolkits
Data Grid
Remote Computation
Remote Vizualization
Collaboratories
Portals
...
Remote Sensors
Grid Services (Middleware)
Protocols, Authentication, Policy, Instrumentation,
Resource Management, Discovery, Events, etc.
Grid Fabric (Resources)
Storage Networks, Computers, Display Devices, etc.
and their associated local services

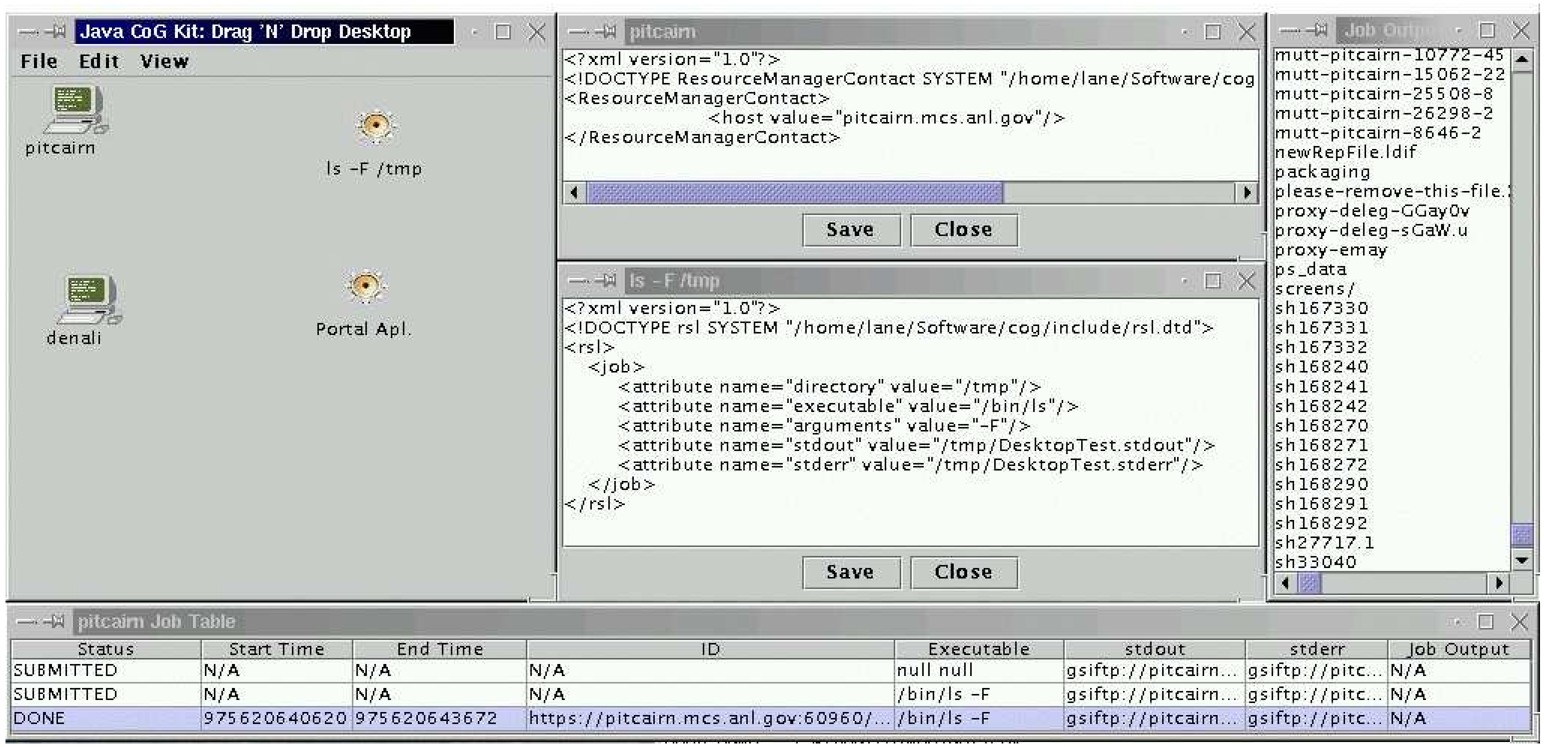

Research
Portal
Financial
Market
Portal
Farmer
Portal
User customized
Portal


Science Portals
Integrated Event Service
Uniform Access Service
Datamining Service
...
Languages
Frameworks
Environments
CoG Kits
Perl Java Jini Corba DCOM
Common Grid Fabric and Services
Security Services
Resource Management Services
Data Management Services
Monitoring Services
Event Services
Collaborative Services
...


Application Level Utilities and GUI components
Low-Level GUI Components
Grid Desktop
MDS browser
MDS searchtable
...
Low-Level Utility Components
Job & JobSets
MDS
CoordinateServer
...
Low-Level Grid Interface Components
RSL
GRAM
GARA
GASS
GSI-FTP
...
Grid Middleware & Fabric
RSL
GRAM
GARA
GASS
GSI-FTP
...


o=Globus, c=US
Browse
Refresh
Set Options
hn gramversiondate gramversion gramsecurity freenodes totalnodes
nene.umiacs.umd.edu 1998/09/1901:13:02 1.15 ssleay 16 16
nowpaci0.cs.berkeley.edu 1998/09/1901:13:02 1.15 ssleay 0 4
oven01.icase.edu 1998/09/1901:13:02 1.15 ssleay 2 2
lorax.ucsd.edu 1998/09/1901:13:02 1.15 ssleay 1 1
ajnl.anu.edu.au N/A N/A N/A N/A N/A
abba.pdc.kth.se 1998/09/1901:13:02 1.15 ssleay 1 1
abba.pdc.kth.se 1998/09/1901:13:02 1.15 ssleay 3 3
lookout.extreme.indian... 1998/09/1901:13:02 1.15 ssleay 0 1
zam361.zam.kfa-juelich.de 1998/09/1901:13:02 1.15 ssleay 1 1
aquila01.spa.lsi.usp.br 1998/09/1901:13:02 1.15 ssleay 1 1
aquila02.spa.lsi.usp.br 1998/09/1901:13:02 1.15 ssleay 1 1
aquila10.spa.lsi.usp.br 1998/09/1901:13:02 1.15 ssleay 1 1
conundrum.ucsd.edu 1998/09/1901:13:02 1.15 ssleay 0 1
tazio.csres.utexas.edu 1998/09/1901:13:02 1.15 ssleay 3 8
deepspace.center.osaka... 1998/12/0717:03:24 1.17 ssleay 1 1
bester.sandia.gov N/A N/A N/A N/A N/A
osprey2.npac.syr.edu 1998/09/1901:13:02 1.15 ssleay 2 2
osprey4.npac.syr.edu 1998/09/1901:13:02 1.15 ssleay 2 2
pdph3.trc.rwcp.or.jp N/A N/A N/A N/A N/A
o=Globus, c=US
Displayed? Attribute
dn
hn
contact
apiversion
apitype
managedresources
gramversion
gramversiondate
gramsecurity
freenodes
totalnodes
scheduledjob
Update


GECCO: Graph Enabled Control COmponent
SnB_71_80
SnB_51_60
SnB_61_70
SnB_41_50
SnB_21_30
SnB_31_40
SnB_11_20
SnB_1_10



Java CoG Kit: Drag 'N' Drop Desktop
File Edit View
pitcairn
ls -F /tmp
denali
Portal Apl.
pitcairn
<?xml version="1.0"?>
<!DOCTYPE ResourceManagerContact SYSTEM "/home/lane/Software/cog
<ResourceManagerContact>
<host value="pitcairn.mcs.anl.gov"/>
</ResourceManagerContact>
Save
Close
ls -F /tmp
<?xml version="1.0"?>
<!DOCTYPE rsl SYSTEM "/home/lane/Software/cog/include/rsl.dtd">
<rsl>
<job>
<attribute name="directory" value="/tmp"/>
<attribute name="executable" value="/bin/ls"/>
<attribute name="arguments" value="-F"/>
<attribute name="stdout" value="/tmp/DesktopTest.stdout"/>
<attribute name="stderr" value="/tmp/DesktopTest.stderr"/>
</job>
</rsl>
Save
Close
mutt-pitcairn-10772-45
mutt-pitcairn-15062-22
mutt-pitcairn-25508-8
mutt-pitcairn-26298-2
mutt-pitcairn-8646-2
newRepFile.ldif
packaging
please-remove-this-file.
proxy-deleg-GGay0v
proxy-deleg-sGaW.u
proxy-emay
ps_data
screens/
sh167330
sh167331
sh167332
sh168240
sh168241
sh168242
sh168270
sh168271
sh168272
sh168290
sh168291
sh168292
sh27717.1
sh33040
pitcairn Job Table
Status | Start Time | End Time | ID | Executable | stdout | stderr | Job Output
SUBMITTED | N/A | N/A | N/A | null null | gsiftp://pitcairn... | gsiftp://pitc... | N/A
SUBMITTED | N/A | N/A | N/A | /bin/ls -F | gsiftp://pitcairn... | gsiftp://pitc... | N/A
DONE | 975620640620 | 975620643672 | https://pitcairn.mcs.anl.gov:60960/... | /bin/ls -F | gsiftp://pitcairn... | gsiftp://pitc... | N/A

B END