Designing Grid-based
Problem Solving Environments and Portals

Gregor von Laszewski, Ian Foster, Jarek Gawor, Peter Lane, Nell Rehn, Mike Russell
Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, U.S.A.
gregor@mcs.anl.gov

Abstract

Building Problem Solving environments in the emerging national-scale Computational Grid infrastructure is a challenging task. Accessing advanced Grid services, such as authentication, remote access to computers, resource management, and directory services, is usually not a simple matter for problem solving environment developers. 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 the Grid and particular commodity frameworks familiar to problem solving environment developers. In this paper, we explain why CoG Kits are important for problem solving environment developers, 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 development of next-generation problem solving environments (PSEs)[12] is influenced by rapid advances in the world of commodity computing and the emerging national-scale Computational Grid. The explosive growth of the Internet and of distributed computing in general has led to significant technology improvements in several domains that are important for the development of PSEs accessing large-scale computational resources. In the world of commodity computing, a broad spectrum of distributed computing technologies (Web protocols, Java, JINI, CORBA, DCOM, 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, 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 have established the *Commodity Grid (CoG) project*, with the twin goals of (a) enabling developers of PSEs to exploit commodity technologies wherever possible and (b) exporting Grid technologies to commodity computing for easy integration in PSEs.

A first activity being undertaken within the CoG project is the design and development of a set of Commodity Grid Toolkits (CoG Kits), that define and implement 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.

As part of these activities, we have successfully developed a Java-based Commodity Grid Toolkit (Java CoG Kit) that defines and implements a set of general components mapping Grid functionality into the Java framework. The Java CoG Kit is of particular interest for PSE developers because it allows them to implement preinstalled heavy-weight applications to be started on user-accessible compute servers, as well as lightweight Web interfaces or portals allowing access to sophisticated remote compute services.

The primary goal of our research is not to build a PSE that will solve a specific problem for a particular application area. Instead, our focus is on developing a software infrastructure to make it easier to build and deploy powerful PSEs. We have based our development of the Java CoG kit on our experiences with application users in various problem domains. Thus, we are confident that the toolkit is general enough to be useful for a large number of PSE developers.

While we have introduced in [17] the general concepts of the Java CoG Kit, we will illustrate in this paper its practical use in the development of problem solving environments. Additionally, we introduce here new components and more sophisticated security concepts that are of particular interest to developers of chemistry problem-solving environments.

2. Portals to Problem Solving Environments

For readers to understand the scope of this work, we explain the terms *problem solving environment* and *portal*, since multiple definitions are used for both terms in the literature.

2.1. Problem Solving Environment

Our understanding of a PSE follows approximately the definition given in [7]: "A problem solving environment is a computational system that provides a complete and convenient set of high level tools for solving problems from a specific domain. The PSE allows users to define and modify problems, choose solution strategies, interact with and manage appropriate hardware and software resources, visualize and analyze results, and record and coordinate extended problem solving tasks. A user communicates with a PSE in the language of the problem, not in the language of a particular operating system, programming language, or network protocol."

For our research focus we assume that the problems must access remote resources, potentially in a secure fashion, and may require a large amount of compute and/or data resources. The process of solving the problem is steered by the scientist, and its progress may be monitored through Internet browsers or special-purpose application-monitoring programs.

2.2. Requirements for PSE Portals

We identified a list of characteristics that influenced our PSE toolkit design [1]:

Problem-oriented. The PSE should allow specialists to concentrate on their discipline, without having to become experts in computer science issues, such as networks, parallel computing, or the World Wide Web.

Integrated. Many problems and their solution strategies are extremely heterogeneous: in models, codes, applications, and machines. A PSE must be designed to manage this heterogeneity in an integrated way, so that the user is presented with a predictable and consistent PSE.

Collaborative. Most science and engineering projects are performed in collaborative mode with physically distributed participants. A PSE must include the ability to foster collaborative solution strategies. We assume that a general-purpose video conferencing tool can be provided with common off-the-shelf tools developed by commercial companies. Nevertheless, it may be necessary to develop special-purpose collaborative tools that are not provided by third parties.

Distributed. Besides the need to support distributed collaboration between scientists, many problems we have been dealing with (such as Grand Challenges) can be solved only while accessing large distributed resources (such as storage and compute resources) in conjunction with each other. A PSE must be able to access these distributed resources seamlessly and in collaboration.

Persistent. Since developing a solution for a problem may require significant time, it is desirable to provide a persistent environment that allows the researcher to resume the solution process at a later time at a potentially different location. Thus, it is necessary to be able to checkpoint not only the state of the calculation but also the state of the PSE user interface. The persistence of a PSE could be enhanced with preferences that are either set by the user or are detected automatically by the PSE. Such functionality could be achieved with the integration of what is called an electronic notebook.

Open, flexible, adaptive. Problem strategies require being able to integrate novel ideas. A sophisticated PSE building tool must be able to tailor or add new functionality within its existent base.

Graphical, visual. The use of graphics and visuals can enhance the usability of the PSE, for example, through animated tables and directed graphs to visualize the state of the application. Furthermore, it must be possible to integrate custom-designed graphical and visual inputs and outputs.

2.3. Portal for Problem Solving Environments

A "Web portal" is commonly defined as an entry point or starting site for the World Wide Web, combining a mixture of content and services that attempts to provide a personalized "home base" for it's audience. Features include customizable start pages to guide users easily through the services provided by the portal. Such services include filterable e-mail, chat rooms and message boards, person-

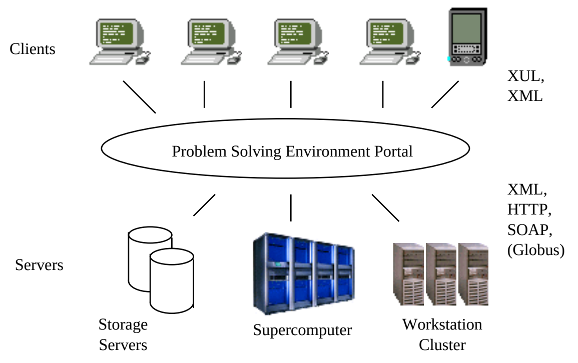


Figure 1. A computing portal interfaces clients with Grid resources such as storage servers, supercomputers, and workstation clusters.

alized news, gaming channels, shopping capabilities, advanced search engines, and personal homepage construction kits. Examples for consumer-oriented portals are provided by AOL and Yahoo.

In this spirit, we suggest that a convenient way of interfacing with a PSE is to design portals for a scientific domain or a particular problem strategy. Besides providing collaborative, interactive, and information services, such portals include also services that are unique for the domain but are typically not provided by consumer-oriented portals. These services include interfaces between users of the PSE with the help of clients ranging from graphics workstations to palm pilots to the resources available as part of the computational Grid (Figure 1). Naturally, not all capabilities of a portal may be exposed by less capable access devices such as palm pilots. Nevertheless, the ability to send a message to a beeper, palm pilot, or cell phone adds significant value to the PSE functionality by notifying the user of the existence of a collaborative session or the completion of a problem solution. Hence, the ability to access a portal with various (even less capable) devices is an integral part of our design.

2.4. Users and Usage Modes of PSE Portals

Portal development for PSEs first requires determining which customer group will be using the portal. We distinguish three target groups:

1. *Novice* science or problem solving environment users, that is, casual or novice users using readily available solutions to problems. The problem strategy is non-transparent to novice users.

2. *Expert* science or problem solving environment users, that is, users in the domain for which the portal is developed. Such users are able to extend the portal while providing solution strategies as used by the novice users or themselves.

3. *Developer* of application or problem solving environments, providing general-purpose components used by experts or novice users.

In addition, we distinguish between interactive and batch mode in which jobs are submitted from the problem solving environment to the backend systems by the users. We have to be able to support the use of compute resources through fine-grained parallel programs, typically provided through MPI message-passing parallel programs, or coarse-grain parallel programs through job dependencies between jobs submitted to the batch processing systems or a fork jobmanager. The toolkit we describe in this paper supports these usage modes.

3. Architecture

Because of the diversified use of a PSE portal, the architecture of such an environment must be flexible. Thus, it is not feasible to develop a point solution for a single problem. Needed instead is a portal toolkit that includes a set of services exposed via APIs that can be used to assemble a point solution for a problem. Figure 2 and Table 1 outline the various groups of services that we initially focus on and that must be integrated into a portal toolkit. Each portal component may have several subcomponents that support the tasks performed as part of the computing portal for problem solving environments. The components in bold text of Figure 2 are developed as part of the CoG Kit. Other components are provided either by commodity software or the application programmers. The flexible design makes it possible to integrate new components into the framework or replace existing modules.

3.1. Grid Core Services

The scientific problem-solving infrastructure of the twenty-first 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 [5][6]. 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. In designing PSE portals, we make extensive use of these technologies, including Globus services, such as

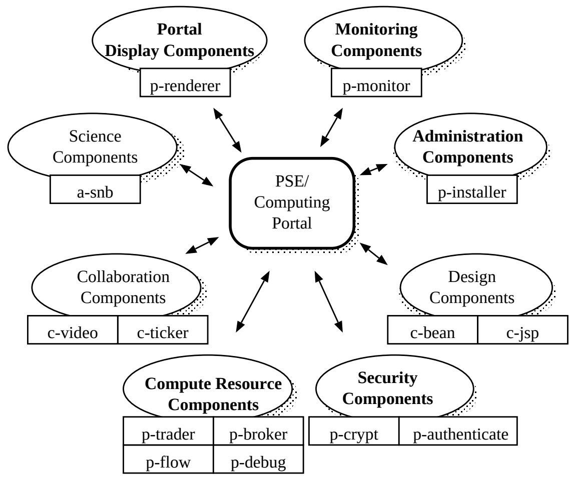


Figure 2. A computing portal is built with the help of a variety of portal components ranging from specialized application- specific portal components to components for using distributed compute resources or other Grid infrastructure.

Table 1. Portal Components

Portal Component	**Sub Component**	**Function**
Collaboration	*c-video*	*Video collaboration (e.g. netmeeting)*
	c-ticker	*news server*
Design	*p-bean*	*Java IDE (e.g. VisualJava, Forte, ..)*
	p-jsp	*Java IDE*
Science	*a-snb*	*Application specific provided by scientists*
Compute Resource	p-trader	locates compute resources
	p-broker	schedules jobs
	p-flow	dependencies between jobs
	p-debug	debugs job execution
	gram	*Globus job submission*
Security	p-crypt	sends secure messages
	p-authenticate	authenticates to the system
	gsi	*Grid Security Infrastructure*
Administration	p-installer	installs software on client
Monitoring	p-monitor	monitors the state
	mds	*Globus Metacomputing Directory Service*
Display	p-renderer	displays information from XML

- the information service (MDS), which enables uniform access to information about the structure and state of Grid resources;
- an authentication and authorization service (GSI), which provides mechanisms for establishing, identifying, and creating delegatable credentials; and
- a uniform job submission service across distributed scheduling systems (GRAM).

These Grid services are often termed “middleware”: they typically involve a distributed state and can be viewed as a natural evolution of the services provided by today’s Internet. They build the basis for developing a Grid-based problem solving environment because many of the portal components use their services.

3.2. Job Submission and Execution

One of the main services a PSE portal must provide is to job submission to remote resources. This must be done in seamless fashion from the desktop with a single sign-on authentication. Computers must be located and the computation must be started on the selected systems. It is essential to monitor the progress of the job execution and obtain the results of the calculation through, for example, output files that may be manipulated locally on the client side (the computer from which the job was initiated). We are able to support such uniform job submission while using the Globus metacomputing toolkit to access Grid resources securely.

Authentication The first step of the job submission is to authenticate with the system. Authentication is the process to verify the identity of an entity. Although the cryptographic algorithms that form the basis of most security systems–such as public key cryptography–are relatively simple, it is a challenging task to use these algorithms to meet diverse security goals in complex, dynamic problem solving environments, with potentially large and dynamic sets of users and resources and fluid relationships between users and resources. Authentication solutions for problem solving environments in a Computational Grids must solve two problems not commonly addressed by standard authentication technologies.

The first problem is support for local heterogeneity. The resources available in the Grid are operated by a diverse range of entities, each defining a different administrative domain.

The second problem support for N-way security contexts. In traditional client-server applications, authentication involves just a single client and a single server. In contrast, a Grid-based PSE may require and dynamically maintained resources. Thus, it must be possible to establish a security relationship between any two processes in the computation used to solve the problem even if they are in different administrative domains. To simplify our task we use the Grid security infrastructure (GSI) that deals with the authentication. GSI policy allows a user to authenticate just once per computation, at which time a credential is generated that allows processes created on behalf of the user to acquire resources, and so no, without additional user intervention. Local heterogeneity is handled by mapping a user's Grid identity into local user identities at each resource. In summary, the GSI security model provides PSEs the following advantages: single sign-on for all resources, no need for user to keep track of accounts and passwords at multiple sites, and no plaintext passwords.

Protocol-based Job Submission Recently, Globus has been enhanced to include an HTTP-based protocol for job submission. Thus, job submission can be initiated from a client on which no other Globus components are installed. Figure 3 shows the Globus components that are involved in such a job submission. First, one has to authenticate with the system, which is done with the help of public key infrastructure and a proxy delegation while generating a temporary key. Jobs are submitted from the client side through API calls known as *gram-submit* and *gram-request*. The gatekeeper on the Globus-enabled resource verifies whether the user is allowed to submit a job to it and checks the availability of the user's public key in a grid map file local to the resource. Once a job has been successfully submitted to the system, it is started with the help of the job manager, and its state is monitored with the help of the reporter. During startup of a job a user can register callback handlers that provide job status updates. In our Java CoG Kit we have implemented all components and services responsible for the proxy initialization and the job submission. Furthermore we have replaced the C-based callback service with a Java-based event service. Thus, all components to submit a job are available in pure Java, allowing even Windows clients to submit jobs to Globus servers.

3.3. Additional Security Issues

In the preceding sections we addressed security issues related to authentication and authorization while using the security policy suggested by Globus. The authorization to use a particular Grid resource can be controlled via a grid-map file and appropriately specified group permissions controlled by the local system administrators.

Nevertheless, we still have to address issues such as the secure communication between programs. To guarantee privacy, we use the security mechanisms provided by secure socket connections, which we can obtain through GlobusIO.

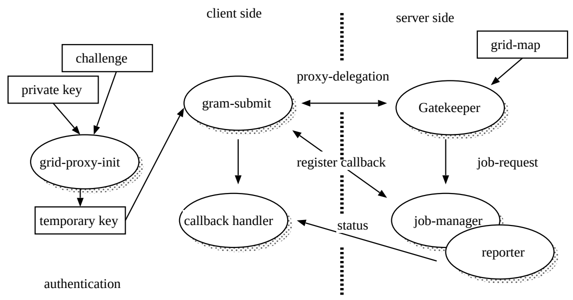


Figure 3. The components of the Globus security infrastructure used during job submission. All client side components are available within the CoG Kit as pure Java components.

This allows us to send messages and data in a secure fashion between compute resources.

4. Java CoG Kit

In the remainder of this paper we focus our attention on our Java CoG Kit prototype, which enables us to build the components listed in Table 1 and used as part of a PSE. 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 the classes that we deem most useful for the development of PSE-based Grid applications. The design of the Java CoG Kit is intended to facilitate the development of future components as a community project. To support an iterative process of definition, development, and application of a Java CoG Kit in collaboration with other teams, we classify components in four layers. This categorization provides the necessary subdivisions to coordinate such a challenging open community software engineering task.

Low-Level Grid Interface Components provide mappings to commonly used Grid services: for example, the *Grid information service* (the Globus Metacomputing Directory Service, MDS), which provides Lightweight Directory Access Protocol (LDAP) [9] 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 and DUROC services); and *data access services*, for example, via the Globus GASS service [3].

Low-Level Utility Components are utility functions designed to be reused by many users. Examples are com-

```
//    Step 0. Initialization
    MDS mds=new MDS("www.globus.org","389","o=Grid");
//Step 1. Search for an available machine
      result = mds.search
     ("(objectclass=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 data and return
//                  the    attributes: server,port,directory,file
      dn = mds.search
       ("(objectclass=MoleculeStructureData)(name=cholera)",
        "dn", MDS.SubtreeScope);
       result = mds.lookup (dn,"server,port,directory,file");
//    Step 2.b) download the data to the machine
      url = result.get("server")+":"+ result.get("port")+":"
          + result.get("directory")+"/"+ result.get("file");
      data = server.fetch (url, machineContact);
//     Step 3. Prepare a description for running the model
      RSL rsl = new RSL("(executable=snb)(processors=64)
             (arguments=-out snb.out)
             (arguments=-in " + data.filename +")");
//    Step 4. Submit the program
     GramJob job = new GramJob();
      job.addJobListener(new GramJobListener() {
        public void stateChanged(GramJob job) {
             // react to job state changes
        }
      });
      try{
           job.request(machineContact, rsl);
      } catch (GramException e) {
          //    problem submitting the job
     }
```

Figure 4. This sample script demonstrates how we access basic Grid services with the help of the Java CoG Kit. Here data for a structural biology code called SnB are located, an appropriate machine is selected, and the calculation is executed on that machine.

ponents that use information service functions 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) or the Globus job submission language (RSL), that locate the geographical coordinates of a compute resource and that test whether a machine is alive.

Low-Level GUI Components provide a basic graphical components that can be reused by application developers. Examples 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 stock market monitor, a graphical climate data display component, or a specialized search engine for climate data.

Figure 4 shows how a small set of services provided by the Java CoG Kit may be used in practice. This Java program skeleton 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 a problem specific algorithm is determined, and the job is executed on that machine, resulting in the generation of an output file.

4.1. Low-Level Grid Interface Components

We describe here 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 describe only the general functionality of these packages. A complete list of the classes and methods accompanies the distribution [18].

RSL The package *org.globus.rsl* provides methods for creating, manipulating, and checking the validity of the Resource Specification Language (RSL) expressions used in Globus [8] to express resource requirements. As shown in Step 3 of Figure 4, 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 Resource Allocation Manager (GRAM) services [8], which allow users to schedule and manage remote computations. The classes and methods distributed 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 4 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, as well as the availability of a sophisticated event model in Java. Our implementation utilizes this event model 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 `GramJobListener` 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).`

MDS The package *org.globus.mds* simplifies access to the Metacomputing Directory Service (MDS) [15], 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 [9] client libraries, including JNDI [10], Netscape SDK [11], and Microsoft SDK [13].

As shown in Step 1 of Figure 4, the parameters to initialize the MDS class are the DNS name of the MDS server, the port number for the connection, and the distinguished name (DN) that specifies the root for a search in the directory tree. A search is performed in Step 2a; 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.

GASS The Global Access to Secondary Storage (GASS) service [3] 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 *get(String from, String to)* copies a remote file to a local file, and the method *put(String from, String to)* copies a local file to a remote location. The *fetch* method used in our example (Figure 4) provides a convenient wrapper and uses internally the previously mentioned *get* method.

4.2. 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 PSEs. 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 CORBA objects. The GECCO application introduced in Section 4.4 uses the Java-based machine and job brokers.

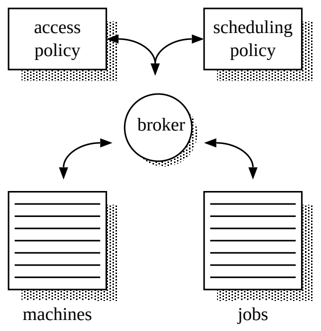


Figure 5. A broker interface allows us to specify an easy way to develop compatible components relying on this interface. Jobs and machines are selected based on a predefined access/security policy as well as a scheduling policy. The policies may be generated dynamically based on other system information.

The broker is a good example of a universally useful component for PSE developers, as well as Grid users. Here a set of jobs and machines is stored in two tables. Dependent on a scheduling and access policy, a machine is selected and a job is scheduled for the execution on this machine (see Figure 5). We have defined a simple interface outlined in Figure 6. This interface allows us to add jobs and machines to the sets so that it is possible to administer them dynamically. With the help of this interface we have defined multiple scheduling policies such as first-come-first-served and load balancing based on resource characteristics. Currently we are investigating the use of economy models for scheduling jobs to machines.

4.3. 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 [17], trees that display the directory information tree of the MDS, and tables to display HBM and network performance data. Each component can be customized and is available as 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.

```
 interface broker ... {
   addJob(JobDescription job)
   deleteJob(JobDescription job)
   addMachine(MachineDescription machine)
   deleteMachine(MachineDescription machine)
   setAccessPolicy(BrokerAccessPolicy policy)
   setSchedulingPolicy(BrokerSchedulingPolicy policy)
      ...
  MachineDescription getMachine()
  JobDescription getJob()
     ...
}
```

Figure 6. This code fragment shows the elementary methods of the broker. Jobs and machines can be added. The job and machine returned by the get methods are defined by the policies and the algorithms defined by an object instantiation of the interface.

4.4. PSE Application Level Utilities and GUI Components

High-level graphical applications combine a variety of CoG Kit components to deliver a single application or applet. These applications can be combined to provide even greater functionality. The user should select the tools that seem appropriate for the task. To demonstrate the range of applications, we have included a set of screen dumps that highlight the look and feel of some applications developed to date.

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 [16][14]. 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 7, each job is represented as a node in the graph. A job is executed as soon as its predecessors are reported to have 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.

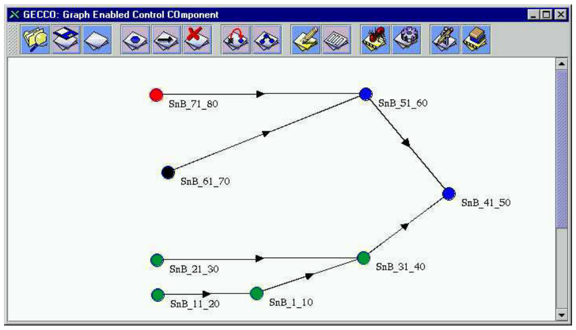
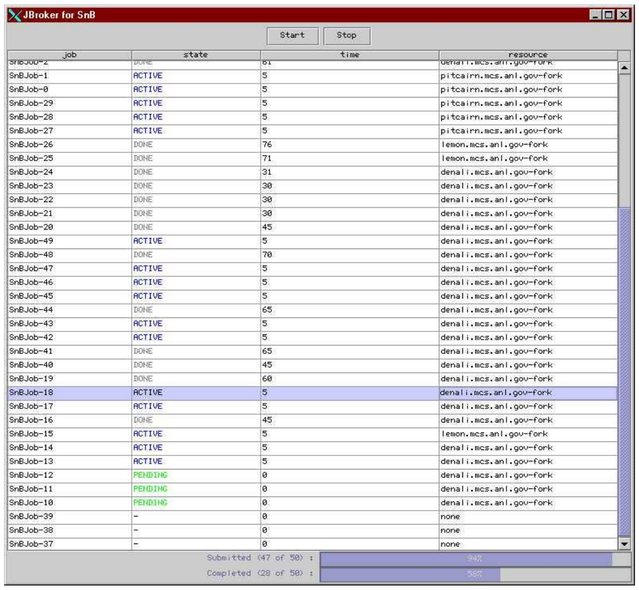


Figure 7. The Grid Enabled Console COmponent (GECCO) allows the user to specify dependencies between tasks that are to be executed in the Grid environment.

High-Throughput Broker We have developed a prototype of a high-throughput broker to test whether the interfaces and classes allow one to easily generate high-level components that simplify job maintenance tasks for certain problem-solving strategies. One of the tasks that has been identified and is common to many solution strategies is to perform a parameter study [2][4]. That is, an algorithm is repeatedly executed with a variety of parameters. Our system is based on the interface of a broker and thus allows us to clearly separate the GUI presentation from the functionality (Figure 8). The prototype looks for compute resources available in a pool of machines formed by a Grid information service with the help of the Globus MDS. From this pool we select those resources that are idle and are available for calculation. If a resource is not able to fulfill a job (because of connection timeout or excessive time needed to complete the job), the resource is automatically removed from the set of viable candidates. The set of resources as well as those removed from the list can be manipulated through an interactive shell. A similar interface exists for the jobs. Special attention has to be placed on the implementation of such a broker. Although it is possible to spawn for each job and machine a thread that maintains the appropriate object, we have chosen to maintain the jobs and machines in lists to avoid the overhead associated with threads and the expected resource limitations on the machine on which the system is running. Thus, we are able to handle submissions that maintain 10,000 or more jobs, a task that would otherwise be impossible.

5. Installation and Upgrading

An important function that must be provided by a PSE is to install and upgrade the software that accesses the various services exposed as part of its design. Using Java will provide us with several options for deploying our client soft-



job	state	time	resource
SnBJob-2	DONE	61	denali.mcs.anl.gov-fork
SnBJob-1	ACTIVE	5	pitcairn.mcs.anl.gov-fork
SnBJob-0	ACTIVE	5	pitcairn.mcs.anl.gov-fork
SnBJob-29	ACTIVE	5	pitcairn.mcs.anl.gov-fork
SnBJob-28	ACTIVE	5	pitcairn.mcs.anl.gov-fork
SnBJob-27	ACTIVE	5	pitcairn.mcs.anl.gov-fork
SnBJob-26	DONE	76	lemon.mcs.anl.gov-fork
SnBJob-25	DONE	71	lemon.mcs.anl.gov-fork
SnBJob-24	DONE	31	denali.mcs.anl.gov-fork
SnBJob-23	DONE	30	denali.mcs.anl.gov-fork
SnBJob-22	DONE	30	denali.mcs.anl.gov-fork
SnBJob-21	DONE	30	denali.mcs.anl.gov-fork
SnBJob-20	DONE	45	denali.mcs.anl.gov-fork
SnBJob-49	ACTIVE	5	denali.mcs.anl.gov-fork
SnBJob-48	DONE	70	denali.mcs.anl.gov-fork
SnBJob-47	ACTIVE	5	denali.mcs.anl.gov-fork
SnBJob-46	ACTIVE	5	denali.mcs.anl.gov-fork
SnBJob-45	ACTIVE	5	denali.mcs.anl.gov-fork
SnBJob-44	DONE	65	denali.mcs.anl.gov-fork
SnBJob-43	ACTIVE	5	denali.mcs.anl.gov-fork
SnBJob-42	ACTIVE	5	denali.mcs.anl.gov-fork
SnBJob-41	DONE	65	denali.mcs.anl.gov-fork
SnBJob-40	DONE	45	denali.mcs.anl.gov-fork
SnBJob-19	DONE	60	denali.mcs.anl.gov-fork
SnBJob-18	ACTIVE	5	denali.mcs.anl.gov-fork
SnBJob-17	ACTIVE	5	denali.mcs.anl.gov-fork
SnBJob-16	DONE	45	denali.mcs.anl.gov-fork
SnBJob-15	ACTIVE	5	lemon.mcs.anl.gov-fork
SnBJob-14	ACTIVE	5	denali.mcs.anl.gov-fork
SnBJob-13	ACTIVE	5	denali.mcs.anl.gov-fork
SnBJob-12	PENDING	0	denali.mcs.anl.gov-fork
SnBJob-11	PENDING	0	denali.mcs.anl.gov-fork
SnBJob-10	PENDING	0	denali.mcs.anl.gov-fork
SnBJob-39	-	0	none
SnBJob-38	-	0	none
SnBJob-37	-	0	none

Submitted (47 of 50) : 94%
Completed (28 of 50) : 56%

Figure 8. A high throughput broker allows the submission of many jobs as part of a problem. After all jobs are completed a solution of the problem can be obtained. The progress of the calculation is monitored with a GUI.

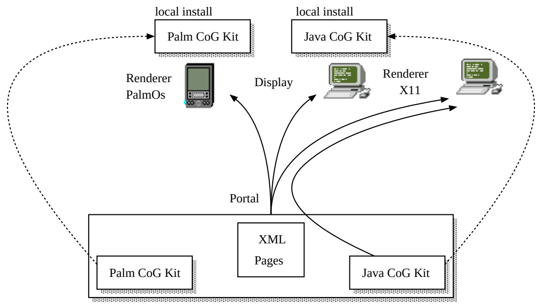


Figure 9. The installation of the CoG Kit onto a client can be done prior to the start of the application as a standalone application or the installation of a library or during an on-demand execution.

ware. 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 9). This strategy is today used by many commercial portals as part of their access software enabled with the help of so-called 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.

6. 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 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 proven 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.

With the help of the CoG Kits we have prototyped a portal to a structural biology problem solving environment. Other projects are currently investigating the use of the CoG Kit to simplify the access to Grid resources. Such projects include the astrophysics portal Cactus, the NCSA Userportal, and SDSC Hotpage. The requirements demanded by such projects have influenced our present design, and we are collaborating with project developers to enhance the components we provide in the CoG Kit. Most recently, we have started to address the integration of components developed by other collaborators.

7. Acknowledgments

This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Globus research and development is supported by DARPA, DOE, and NSF. We thank Geoffrey C. Fox, Dennis Gannon, and Jason Novotny for valuable discussions during the course of the CoG Kit development. This work would not have been possible without the help of the Globus team.

For up-to-date release notes, and further information readers should refer to the Web page http://www.globus.org/cog [18].

References

[1] Marc Abrams, Donald Allison, Dennis Kafura, Calvin Ribbens, Mary Beth Rosson, Clifford Shaffer, and Layne Watson. PSE Research at Virginia Tech: An Overview. Department of Computer Science, Virginia Tech, Blacksburg, VA 24061, 1999.

[2] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed workstations. In *Proc. 4th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1995.

[3] Joseph Bester, Ian Foster, Carl Kesselman, Jean Tedesco, and Steven Tuecke. GASS: A data movement and access service for wide area computing systems. In *Proc. IOPADS'99*. ACM Press, 1999.

[4] D. H. J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A Worldwide Flock of Condors: Load Sharing among Workstation Clusters. *Future Generation Computer Systems*, 12, 1996.

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

[6] Ian 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.

[7] E. Gallopoulos, E. Houstis, and J.R. Rice. Problem-Solving Environments for Computational Science. *IEEE Computational Science and Engineering*, 1:11–23, 1994.

[8] The Globus GRAM. http://www.globus.org/gram.

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

[10] JAVA Naming and Directory Interface (JNDI). http://java.sun.com/products/jndi. Version 1.2.

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

[12] J. R. Rice and R. F. Boisvert. From scientific software libraries to problem-solving environments. *IEEE Computational Science and Engineering*, Fall:44–53, 1996.

[13] Richard Schwartz. *Windows 2000 : Active Directory Survival Guide*. John Wiley and Sons, 1999.

[14] Gregor von Laszewski. A Loosely Coupled Metacomputer: Cooperating Job Submissions across Multiple Supercomputing Sites. *Concurency, Experience, and Practice*, Mar. 2000.

[15] Gregor 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.

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

[17] Gregor von Laszewski, Ian Foster, Jarek Gawor, Warren Smith, and Steve Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *ACM 2000 Java Grande Conference*, San Francisco, California, June 3-4, 2000. http://www.extreme.indiana.edu/java00.

[18] Gregor von Laszewski, Jarek Gawor, and Peter Lane. Java CoG Distribution. http://www.globus.org/cog, January 2000. Version 0.8.6.