CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids

Gregor von Laszewski
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439, U.S.A.
gregor@mcs.anl.gov

Ian Foster
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439, U.S.A.
foster@mcs.anl.gov

Jarek Gawor
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439, U.S.A.
gawor@mcs.anl.gov

ABSTRACT
Emerging national-scale "Computational Grid" infrastructures are deploying advanced services beyond those taken for granted in today's Internet: for example, authentication, remote access to computers, resource management, and directory services. The availability of these services represents both an opportunity and a challenge for the application developer: an opportunity because they enable access to remote resources in new ways, a challenge because these services may not be compatible with the commodity distributed-computing technologies used for application development. 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.

Categories and Subject Descriptors
D.1.3 [**Software Engineering**]: Concurrent Programming—*Distributed programming*; D.2.6 [**Software Engineering**]: Programming Environments—*Graphical environments*; D.2.6 [**Software Engineering**]: Programming Environments—*Programmer workbench*

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 [16], Java [14], JINI [1], CORBA [4], DCOM [20], 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* [7], 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), 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.

Our initial focus of our work in this area is on a Java CoG Kit. (We have also started some investigations of Web/CGI, CORBA, and Python CoG Kits.) In the rest of this article, we first review briefly some Grid technologies, then use an example to illustrate what capabilities we want the Java CoG Kit to provide, and finally present technical details on the Java CoG Kit design.

2. GRIDS AND GRID TECHNOLOGIES

The scientific problem-solving infrastructure of the next century will support the coordinated use of numerous distributed heterogenous 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 [7]. 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 [6], 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 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 identify, 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).

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 requires the ability to access compute resources with significant

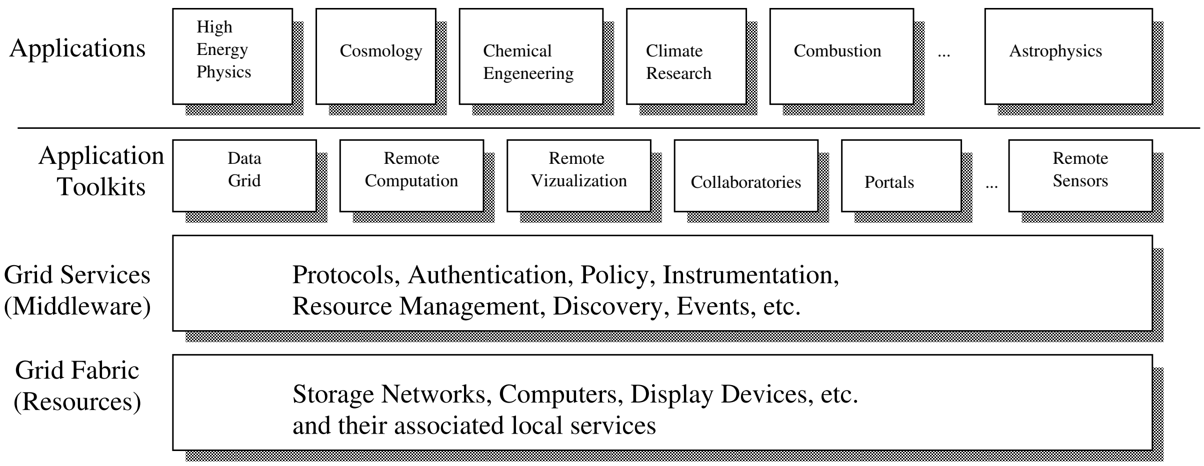

Figure 1: The integrated Grid architecture has four main categories.

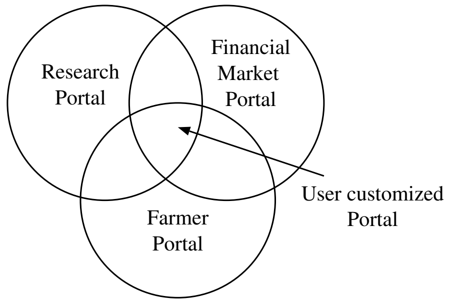

Figure 2: Multiple portals provide access to overlapping functionality, with a particular portal specialized to the requirements of its user.

computational resources. 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; we would also like to be able to enforce *performance guarantees* for data transfers and computations.

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 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 effort by thought to get right. 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. Particular,

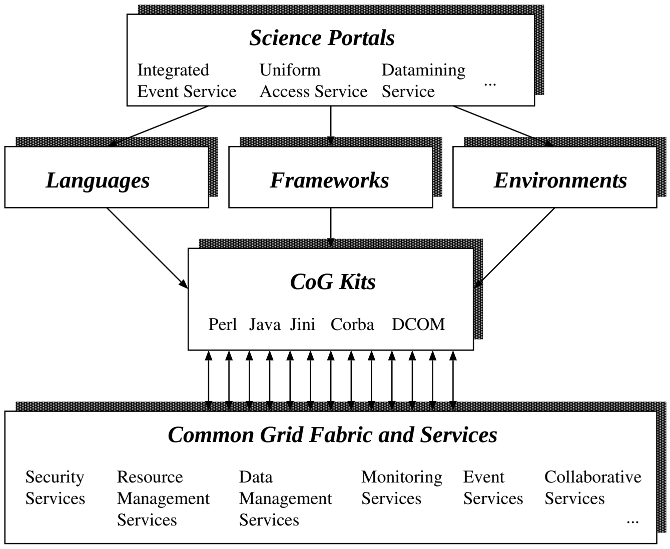

Figure 3: CoG Kits provide a mapping between computing languages, frameworks, and environments and Grid services and fabric. Together Grid services, languages, frameworks, and environments build a powerful development tool for building grid-enhanced applications.

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, and the Distributed Component Object Model (DCOM, CORBA).

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. The design of the Java CoG Kit intends 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 as depicted in Figure 4. This categorization provides the necessary subdivision in order 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) [15] 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 [2].

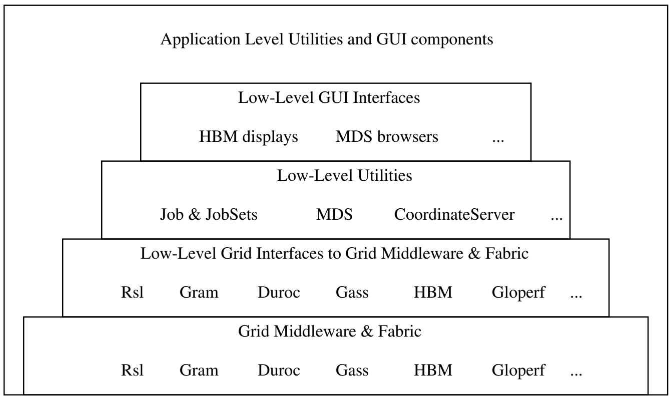

Figure 4: Applications and more complex components can be built with the help of the CoG Kit. Components are classified here based on their role.

Low-Level Utility Components are utility functions designed to be reused by many users. Examples are components 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) [13] or the Globus job submission language (RSL); that locate the geographical coordinates of a compute resource; or that test whether a machine is alive.

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 mentioned 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 forms part of a Climate Portal; it 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 [12] format, a well-known format for storing three-dimensional climate related data. Throughout the remainder of paper we will expand this example as we introduce various Java CoG Kit components.

6.1 Low-Level Grid Mappings

```
// 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 climate data and return
// attributes: server,port,directory,file
dn = mds.search
    ("(objectclass=ClimateData)(year=1999)
       (region=midwest","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=climateModel)
    (processors=64)
    (arguments=-grads)(arguments=-out map.grads)
    (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 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.

In the 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 [27].

RSL. The package *org.globus.rsl* provides methods for creating, manipulating, and checking the validity of the RSL expressions used in Globus [11] to express resource requirements. As shown in Step 3 of Figure 5, the arguments to a *new* call to 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 [10], 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 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, but also because of 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)`.

DUROC. The package *org.globus.duroc* enables a user to coalocate multiple resources. The mapping of the application Globus duroc interface to a Java based event model is similar to that of the *gram* package. In contrast to Gram it allows the programmer to create and monitor multirequest jobs [5]: that is jobs that require the allocation of multiple resources and the creation of management of multi-component tasks.

MDS. The package *org.globus.mds* simplifies the access to the Metacomputing Directory Service (MDS) [25], 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 [15] client libraries, including JNDI [17], Netscape SDK [18], and Microsoft SDK [21].

As shown in Step 1 of Figure 5, 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 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

```
// Step 0: Initialize Heartbeat Monitor Object
HBM hbm = new HBM ("hbm.globus.org", 2222);
hbm.update();

// Step 1: Retrieve name of machine to be watched
machinename = machineContact.hostname();

// Step 2: Get data associated with Client
// Alternative A: Monitor the machine
ClientData cd = hbm.get(machinename);

// Step 3: Evaluate the client data
if (cd.getStatus() != HBM.ACTIVE)
    <deal with the problem>

// Step 2 can be replaced with:
// Alternative B: Monitor the process
ClientData cd = hbm.search(
    machineContact.hostname(),
    "(name=climateModel)");
```

Figure 6: Using the HBM to monitor the progress of a computation. In Alternative A, we check the status of a machine; in Alternative B, we check whether the program with the name `climateModel` is still running. This code might be run in conjunction with Figure 5.

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 [2] 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.glass* 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 5) provides a convenient wrapper and uses internally the previously mentioned *get* method.

HBM. The Globus Heartbeat Monitor (HBM) [22] provides a simple, highly reliable mechanism for obtaining the health and status information of Grid resources. This includes monitoring the state of machines and processes in the Grid. The package *org.globus.hbm* provides classes and methods to conveniently access this service, as illustrated in Figure 6. In alternative A the status of a machine is checked, and if the state is not *active*, an appropriate action is performed. If the status of a process that is registered with the HBM is monitored, alternative B gives an example.

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

```
// 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 8: The program shows the ease of use of the Graphical User Interface for selecting a Grid contact string. (compare Figure 7).

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 utilized 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 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 7 and 8), 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.

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. 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 [26][24]. 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.

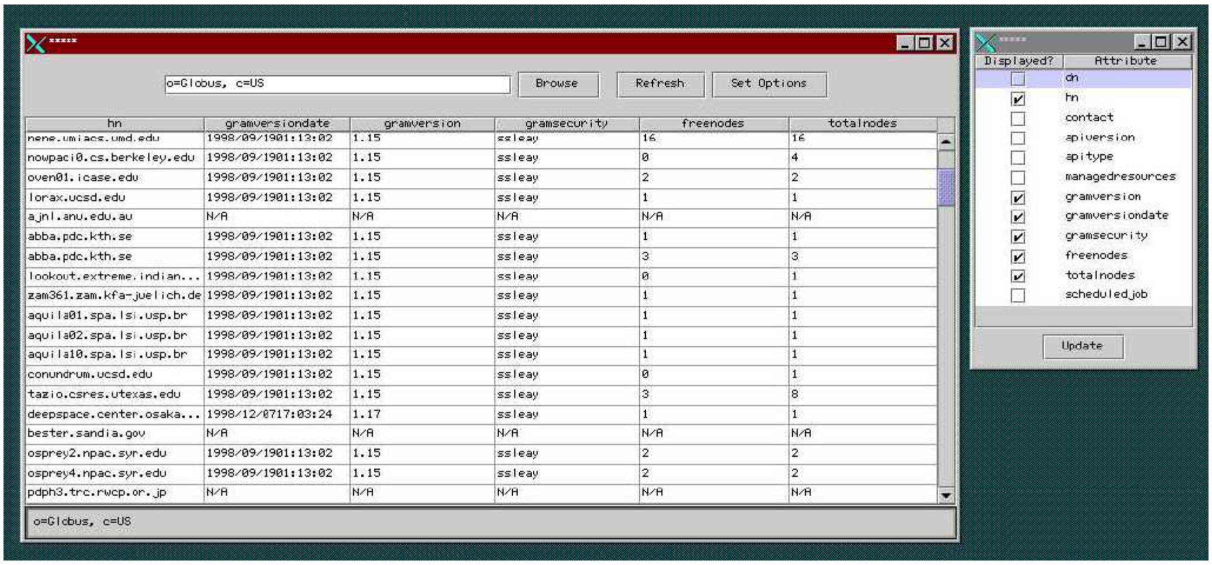

Figure 7: The MDS search table can be used to display selected MDS information in a tabular form. The search string can be specified, and attributes can be selected easily to customize the table.

As shown in Figure 9 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.

GRC. A second example of a high-level application component is an interactive Graphical Resource Co-allocator (GRC) illustrated in Figure 10 [5]. This Java application allows the user to build a network representing the resources required for an application and to describe how the resources should be used. A combination of automatic and manual techniques is then used to guide resource selection, eventually generating an RSL specification for the resources in question. MDS services are used to automatically find candidate sets of resources that meet the user's constraints. The user then manually selects one of the resource sets or requests a further search for candidates. Once the user finds a suitable set of resources, the GRAM or DUROC client libraries are used to execute, monitor, and possibly terminate the application(s) (compare Figure 10).

7. 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 constitutes 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 [9] and Webflow [28].

The latest Globus system to relies in many cases on the HTTP protocol, hence it is possible to integrate such a thin-client as part of a Web browser to allow submission through web pages. Projects like WebSubmit [19] and Hotpage [23] will profit from this change. Making some components available as Java Beans and integrating them into common of-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 in commodity technology in future.

8. 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.

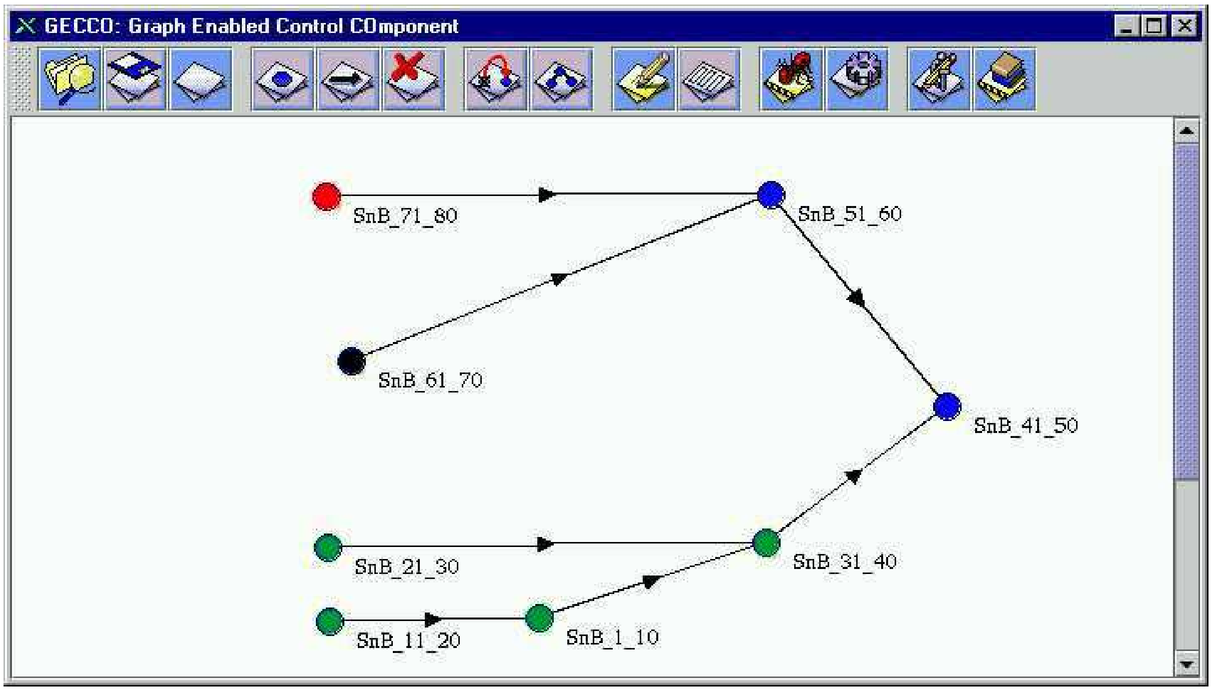

Figure 9: The Grid Enabled Console COmponent (GECCO) allows the user to specify dependencies between tasks that are to be executed in the Grid environment. Here we show a graph created for a crystallography application *Shake 'n Bake*.

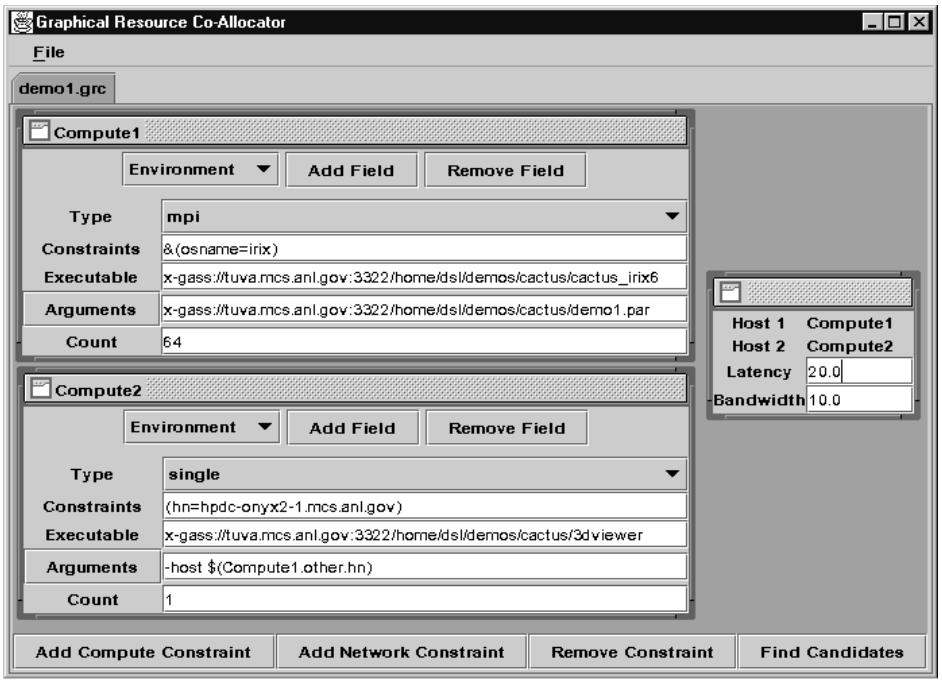

Figure 10: The GRC allows to select a compute resource for scheduling a job interactively from a set of automatically derived machines that fulfill a user-specific constraint.

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.

9. AVAILABILITY

The Java Cog Kit is available in alpha release form the CoG Kit Web pages [27]. 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. We expect that this package will be released as soon as the Globus toolkit API for this area is frozen. For more release notes, we refer to the Web page http://www.globus.org/cog.

10. 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 [3]. We are grateful to members of the NCSA Alliance for enlightening discussions on these topics; in particular, we thank Jay Alameda, Dennis Gannon, Geoffrey C. Fox [8], 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.

11. ADDITIONAL AUTHORS

Warren Smith, and Steve Tuecke

12. REFERENCES

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

[2] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A data movement and access service for wide area computing systems. In *Proc. IOPADS'99*. ACM Press, 1999.

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

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

[5] K. Czajkowski, I. Foster, and C. Kesselman. Co-allocation services for computational grids. In *Proc. 8th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1999.

[6] 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.

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

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

[9] The Gateway Project Web Page. http://www.osc.edu/ kenf/theGateway, 1999.

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

[11] The Globus Resource Specification Language. http://www.globus.org/rsl.

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

[13] I. S. Grah and L. Quin. *XML Specificaton Guide*. Wiley, 1999.

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

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

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

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

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

[19] R. McCormack, J. Koontz, and J. Devaney. Seamless Computing with WebSubmit. *Concurrency: Practice and Experience*, in press.

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

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

[22] P. Stelling, I. Foster, C. Kesselman, C.Lee, and G. von Laszewski. A Fault Detection Service for Wide Area Distributed Computations. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, pages pp. 268–278, July 1998.

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

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

[25] 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 pp. 365–375, 1997.

[26] G. von Laszewski and I. 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.

[27] G. von Laszewski, J. Gawor, and P. Lane. Java CoG Distribution. http://www.globus.org/cog, Jan. 2000. Version 0.8.6.

[28] The Webflow Web page. http://www.npac.syr.edu/users/haupt/WebFlow/demo.html.