An Overview of Grid File Transfer Patterns and Their Implementation in the Java CoG Kit

Gregor von Laszewski,[1,*] Jarek Gawor,[1] Pawel Plaszczak,[1]
Mike Hategan,[1] Kaizar Amin,[1] Ravi Madduri,[1] Scott Gose[1]

[1]Mathematics and Computer Science Division,
Argonne National Laboratory, Argonne, IL 60439
[*] gregor@mcs.anl.gov

Abstract

Accessing files on remote resources is a required function in Grids. In this paper, we report on the file transfer patterns supported in the Java CoG Kit. These patterns are supported by a rich set of accompanying components, including Java classes and methods, command line tools, graphical user interfaces, and portals. The patterns and their implementations are exposed through familiar Java language capabilities of interfaces, hence hiding the underlying protocols. Using these interfaces, one can provide a variety of implementations for diverse file transfer mechanisms and protocols. Together, these tools can be used to implement more sophisticated services. We present a number of prototype applications reusing the Java CoG Kits file transfer patterns. Additionally, we present performance numbers based on a typical client deployment scenario.

1. INTRODUCTION

The Grid approach (von Laszewski & Amin 2004) provides a *vision* to develop an environment for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations under quality of service constraints (Berman, Fox & Hey 2003, Foster & Kesselman 2003).

To support such a vision, a number of important use patterns need to be identified and implementation solutions provided. Some of the most elementary patterns include the execution of programs, the transfer of files, and the authentication and authorization of Grid resources. However, all of these patterns can be abstracted as part of a *general* task pattern with control flow dependencies between each other (von Laszewski 1996). Hence, we can formulate simple task-based workflows within the Grid. With such powerful abstract patterns, application designers can identify reusable abstractions within their applications and apply proven implementation strategies to their own software engineering design, hence reducing the overhead to

create a solution to their specific problem.

In this paper, we focus on issues related to the patterns of file transfer. A frequent requirement of advanced Grid applications is the *access* and *transfer* of files and directories to and from remote resources. Many technical aspects must be considered while dealing with file transfers in Grids. These aspects include the supported security contexts, transfer protocols, class and object hierarchies to support convenient programming, integration with component technologies such as promoted by Java and Enterprise Beans as part of J2EE, and the exposure to Web and Grid services that can be accessed through convenient graphical user interfaces. For full coverage of the most elementary patterns and their implementation, one also must consider the different Grid user communities (von Laszewski, Blau, Bletzinger, Gawor, Lane, Martin & Russell 2002). These communities include common Grid users, who are interested in easy-to-use interfaces; Grid developers, who are interested in easy-to-use APIs and shell commands; and Grid administrators, who are interested in observing the performances of transfers to identify bottlenecks. Hence, a file transfer strategy must be developed and employed in ordinance with the *Gestalt of the Grid* that explicitly targets a particular community (von Laszewski & Wagstrom 2004).

In todays software development process we have to recognize a nonlinear dependency between the technologies used and the potential architecture that is derived from the reuse of diverse methodologies. Although technologies such as Web services have recently received much attention, we must remind ourselves that, in order to provide the exposed functionality, it is not sufficient to express services syntactically and semantically. Indeed, we must provide a *supporting infrastructure to implement* the required functionality. Hence, while focusing on implementations in Java, we are need to develop and support reusable object and data structures and components that simplify the implementation in the object- and component-based Java framework.. The interplay of the different methodologies that we follow in our software engineering approach in regards to developing file transfer solutions is depicted in Figure 1. Ideally we will provide solutions in each of the methodologies.

In the rest of the paper, we present more details of the Grid file transfer patterns and components. All of these solutions are based on the Java CoG Kit GridFTP libraries (von Laszewski, Foster, Gawor & Lane 2001) that provide a convenient API to the GridFTP protocol. First, we define the elementary terminology and introduce the GridFTP protocol. We than discuss the Java CoG Kit API to access the GridFTP protocol. Next, we present a simple command line interface that can transfer files through multiple protocols. We describe a variety of applications that reuse the Java CoG Kit to provide highlevel interfaces and services to Grid file transfers. These include Swing applications, portals, workflow engines, and command line tools. We

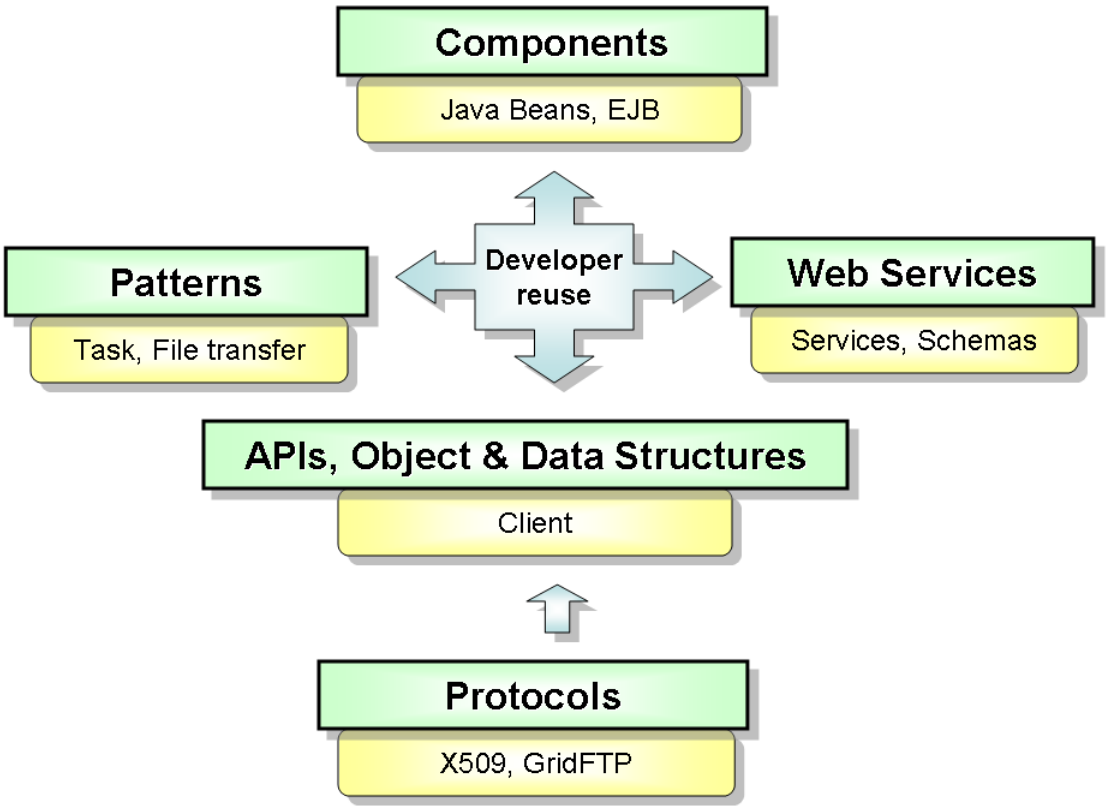

Figure 1: Development of Grid middleware is supported by a variety of software engineering methodologies that encourage reuse.

show how file transfers are integrated easily into a workflow framework. We also show how file transfer is enhanced as part of a Grid service to provide reliability. We conclude the paper with a simple performance comparison of some of our patterns.

2. FILE TRANSFER PROTOCOL AND GRIDFTP

File transfer among remote resources in distributed environments is a common problem in distributed and Internet computing. The concepts of virtualization, security in virtual organizations, and quality-of-service assertions needed as part of complex Grid applications add new dimensions to this problem. In developing a solution, one naturally considers the use and appropriate modification of a protocol that is very well established in the distributed and Internet computing communities. One of the most well known protocols used is the file transfer protocol (FTP) (Postel & Reynolds n.d.). A key characteristic of FTP is the separation of control from data flow through control and data channels. This separation is of especial importance to enable third-party transfers which is a key pattern of Grid computing.

To adapt this protocol to the needs of the Grid, the Globus Alliance selected a set

of protocol features and extensions defined in the RFCs and added a few additional features. As a result, the Globus Alliance proposed GridFTP as a secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks (Allcock, Bester, Bresnahan, Meder & Tuecke 2003). GridFTP provides the following features.

Security. The security mechanism used in GridFTP is based currently on the Grid Security Infrastructure (GSI) (Foster, Kesselman, Tsudik & Tuecke 1998). This enables single sign-on and delegation in a Grid environment that uses GSI as part of the virtual organization deployment infrastructure. GSI is used by GridFTP to authenticate data and control channels.

Performance. To increase the performance, GridFTP provides multiple mechanisms. These include reusable data channels to eliminate the high startup costs during the secure authentication, but most important multiple data channels for parallel file transfers. Additionally, command pipelining can provide an increase in the use of the streams. In case only part of the file is needed on the remote machine, partial file transfers are allowed in either block or striped mode.

Fault support. To enable faulttolerant services, the partial file transfer can be used together with a restart strategy.

Usability. To enable usability, we must support transfers between clients and servers, as well as between servers and servers.

Next we describe some of the most frequently used patterns supported by the GridFTP protocol while focusing especially on second-party, third-party, and sender-receiver modes. We begin by defining the terms second- and third party-transfers. Let us assume a file transfer is defined as the process of transferring a file from a Grid resource A to a Grid resource B. Then, we distinguish second-party transfers that describe transfers between a client and a server; and third party transfers that copies files between two servers under the control of a client.

2.1. Second-party Transfer

A typical implementation of a use pattern to establish second party transfer between a client and a server is as follows:

1. The client establishes a connection with the server through the control channel and transmits transfer parameters and requests the creation of the data channel.
2. After the data channel is established, the client initiates the transfer by sending the transfer command via the control channel.
3. The data is sent through the data channel. The direction of the transfer is determined through the control commands used in Step 1.

2.2. Third-Party Transfer

In second-party transfers, the client initiates the connection. FTP and GridFTP also provide a passive mode (PASV) that switches the client to listen for messages from the server established through a given port (PORT). Hence, it is possible to switch the role between client and server. This means that the active side can be sending data to passive side and vice versa. Thus, being a sender or a receiver is orthogonal to being active or passive, and both of these are separate from the notion of client and server.

The passive mode is especially useful to establish third-party transfers, where one of the servers has to actively initiate the data channel connection. In this case, the client sends PASV to one server (that will be passive) and PORT to the other server (that will be active). Once the data channel has been established the client no longer has to participate in the transfer between the two servers.

2.3. Parallel Transfers

GridFTP replicates the wellknown concept used by some commodity file transfer tools to boost the performance of file transfer: namely, parallel data channels. To implement this pattern, the PORT protocol has been enhanced to include the number of data channels between sender and receiver.

Although the overall concept is intuitive, it can provide increase in performance if the link between sender and receiver is overprovisioned. The reason we see performance gains in GridFTP while using parallel data channels that GridFTP is designed to exploit the TCP protocol. The TCP socket throughput is limited by the TCP window and the related TCP buffer size, which is typically set to 64 KB. As this number is often too small to utilize the maximum available throughput, this overprovisioning can be exploited. However, if everyone on the network exercises this strategy, no room for dealing with network spikes is provided; thus the reliability of the network in regards for large user community may decrease under heavy load.

The benefit of parallel file transfer can be observed in the case of short network outages, software problems, or even router designs that result in dropped packets. As TCPs algorithm calls for a slow startup in such a case, the hope is to avoid this situation through multiple streams. The adverse effect that the use of parallel streams will impose with increased popularity and availability, however, it obvious that we must develop a quality-of-service solution based on service level agreements.

2.4. Striped Transfer

GridFTP introduces striping to transfer data in parallel among multiple senders and receivers, instead of just between one sender and one receiver. The performance in this

case is limited by the performance available in the network between the distributed machines. Naturally, if they share the same physical network, we have not gained any performance increase in comparison to a parallel transfer.

2.5. Transfer Modes

FTP specifies how the data can be marshaled and demarshaled to and from data channels. In order to control this behavior, a transfer mode can be used. In the simplest case, data is pushed to the data channel as a continuous stream (stream mode). Alternatively, data can be sent out in portions or blocks (block mode). GridFTP defines an extended block mode (mode E). In contrast to the block mode from FTP, the block header contains additional information that define the offset from the block's position in the original file. Mode E is the key concept in GridFTP, since parallelism and striping depend on it. Currently, the GridFTP protocol gives a choice between two layouts of the data: *partitioned*, in which the file is divided into large sections and each stripe is given one of them; and *blocked*, in which the file is divided into small portions that are dispatched to the stripes in round-robin fashion.

Several limitations of the protocol make it clear that future Grid-based solutions need to modify the current protocol. These issues are discussed in more detail in (Plaszczak, Link, Wellner & Hubbard n.d.).

3. GRID FILE TRANSFER APIS AND OBJECT STRUCTURES

The Java client API as part of the Java CoG Kit has been available since 1999 and has been continuously updated to support the most useful features of the GridFTP protocol. It implements the following features: file storage and retrieval to and from FTP servers (client-server transfer), third-party transfer, ASCII and IMAGE data types, file data structure, nonprint format control, stream transmission mode, operation in passive and active server mode, parallel transfers, striped transfers, restart markers, and performance markers, GSI security, and certificate revocation lists as part of GSI. A detailed set of documentation can found on the Java CoG Kit Web pages (*Java CoG Kit* n.d.).

3.1. Java-Based Grid Access to Secondary Storage

The Globus Toolkit contains also libraries and command line tools that integrate a variety of protocols, such as FTP, GridFTP, and HTTP, as well as access to local file I/O to enable secure transfers using any combination of these protocols. The Java CoG Kit replicates a subset of this functionality (it does not implement the GASS cache). The ease of use of this API is shown in Figure 2, in which we copy a file between to locations specified as URLs. Writing is allowed only on GridFTP

servers and the local filesystem. Read is allowed on protocols such as FTP, HTTP, and the local filesystem. The Java CoG Kit contains also a command line tool called *globus-url-copy* for convenient use.

```
import org.globus.io.urlcopy.*;
UrlCopy copy = new UrlCopy();
// set the source and destination
copy.setSourceUrl(from);
copy.setDestinationUrl(to);
// set up the transfer mode
copy.setUseThirdPartyCopy(true);
// register a transfer listener
copy.setListener(new UrlCopyListener() {
    public void transfer(int total, int current) {
        System.out.println(total + " " + current);
    }
    public void transferError(Exception e) {
        System.out.println("transfer failed: " + e.getMessage());
     }
});
// start the copy
copy.run();
```

Figure 2: UrlCopy is an easy interface to copying files between a client and a server.

3.2. Java-Based GridFTP Client Library

The Globus Toolkit contains GridFTP servers and clients on a C-based implementation that enhances wu-ftp, a popular FTP server package from Washington University. This implementation supports the majority of the GridFTP protocol features (GSI security, parallel transfer, third-party transfer, partial file transfer). Additionally, the implementation contains a nonprotocol-based enhancement of functionality that allows to add software *plug-ins*. These server side plug-ins allow developers to add customized reliability and fault tolerance, performance monitoring, and extended data processing.

Additionally, the Globus Toolkit contains through the Java CoG Kit a Java-based client library with abstractions that make use of most of the features of the Grid FTP servers. Exceptions are the use of multiple channels in second-party mode between client and servers, which can be easily made available through the Java thread model, and the possibility of embedding plug-ins, which are C-language specific and thus

should be used with caution. The availability of a clientside implementation in pure Java has resulted in an increased stability of the GridFTP C implementation. During the past few years several security and protocol issues were uncovered by the Java CoG Kit team, as it served as independent verification.

Following is an example showing a third-party transfer between two GridFTP servers, namely, hot.mcs.anl.gov and cold.mcs.anl.gov. The former is assumed to be the source of the file, and the latter is assumed to be the destination.

```
import org.globus.ftp.*;
// Move file from cold to hot
GridFTPClient cold = new GridFTPClient("cold.mcs.anl.gov", 2811);
cold.authenticate(null);
cold.setType(Session.TYPE_IMAGE);

GridFTPClient hot = new GridFTPClient("hot.mcs.anl.gov", 2811);
hot.authenticate(null);
hot.setType(Session.TYPE_IMAGE);

// Set the receiving server to passive mode
HostPort hp = cold.setPassive();
hot.setActive(hp);

// Transfer a file.

String remoteSrcFile = "/home/vonLaszewski/sourceFile.txt";
String remoteDstFile = "/home/gvl/destinationFile.txt";

hot.transfer (remoteSrcFile,cold,remoteDstFile,false,null);

// Close both the servers.
hot.close();
cold.close();
```

Figure 3: The Java CoG Kit GridFTP library allows direct control over file transfer parameters.

4. APPLICATIONS

A number of sophisticated applications have been developed based on the Java CoG Kit libraries. These applications include portals and standalone applications. We found that applets are usually unsuited for our purpose and have a high startup cost. In the applications we have chosen, several emphasize a particular aspect such as reliability, user friendliness, and ease of deployment (see Table 1).

Table 1: The various examples using the Java CoG Kit patterns for file transfer emphasize particular requirements

Application Name	Application Type	Section	Emphasis on
OGCE Portal	Portal	4.1	deployment
RFT	Service	4.2	reliability
FTP GUI	Application/Service	4.3	familiarity
Entrada	Application/Service	4.4	deployment, reuse
GridAnt	Application/Service	4.5	workflow

4.1. Portals

As the use of Grid technologies expands and more organizations establish Grids, the need for user-friendly access to Grids becomes critical. Portals provide access to Grid technologies through sharable and reusable components for Web-based access to scientific and business-oriented applications. Sharable components allow the portal developer to quickly create Grid portals from provided libraries that support baseline Grid technologies (such as file transfer, job launching an monitoring, and access to information services), freeing the developers to concentrate on the specialized needs of a particular scientific community or collaboratory.

In fall 2003, the Open Grid Computing Environment (OGCE) project was established to foster collaborations and sharable components with portal developers worldwide (Gannon, Fox, Pierce, Plale, von Laszewski, Severance, Hardin, Alameda, Thomas & Boisseau 2003). Tasks include the establishment of a Grid Portal Collaboratory, a repository of portlet and portal service components, an online forum for developers of Grid portals, and the building of reusable portal components that can be integrated in a common portal container system. OGCE leverages ongoing portals research and development from Argonne National Laboratory, Indiana University, the University of Michigan, the National Center for Supercomputing Applications, and the Texas Advanced Computing Center. Collectively, these institutions form the charter members of the OGCE consortium.

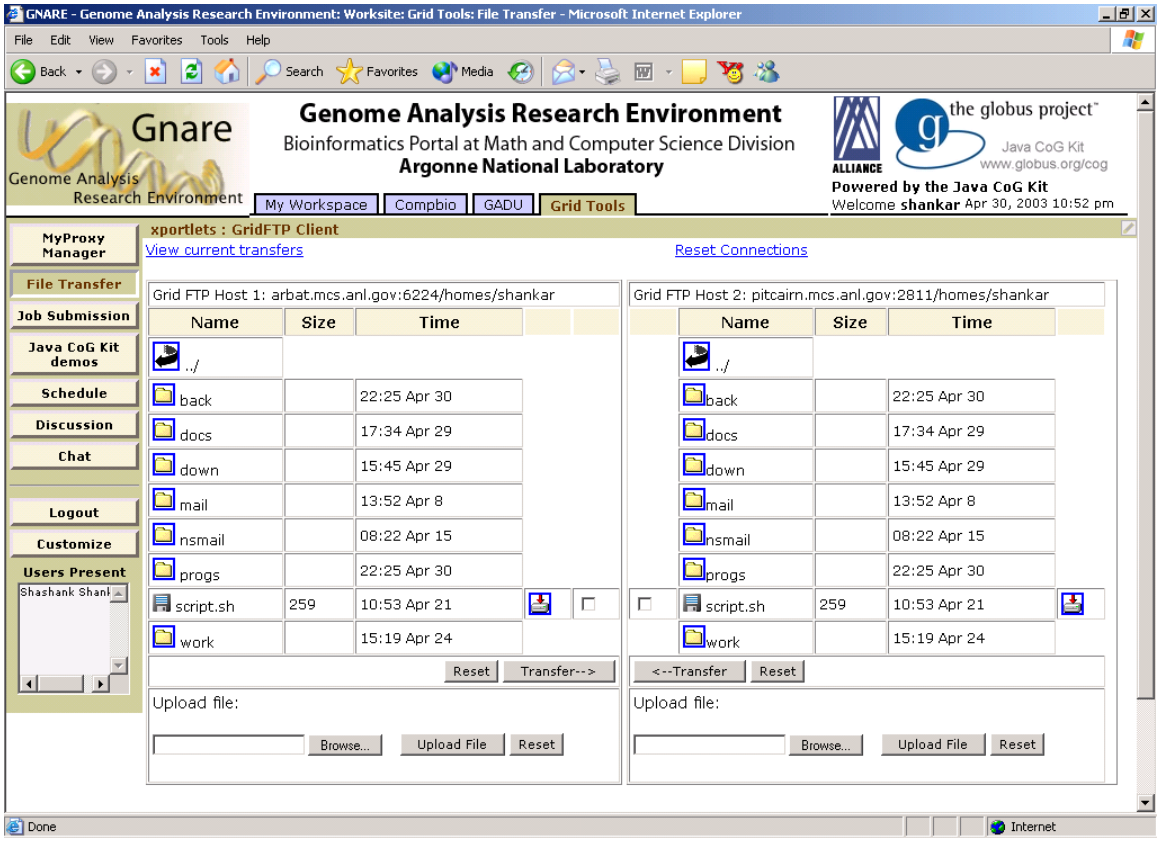

Figure 4: File transfer portlet as reused within a genome analysis portal.

Part of this effort also includes a Grid file transfer portal that can be readily integrated into jetspeed, which is the basis of the portal solution currently developed by this group. The advantage of using a jetspeed-based solution is that each user can customize the panels and components known as portlets that are displayed on personal basis. Figure 4 shows a screenshot of an early version of this portlet. Portlets for file transfer and job submission are currently also available within the Java CoG Kit. Other JSP-based solutions such as GPDK also depend on the Java CoG Kit (Novotny 2003) but are superseded by OGCE.

4.2. Reliable File Transfer Service

The Reliable File Transfer (RFT) service is a Grid service, following the OGSA proposed standard, that provides clients with the ability to control and monitor third-party file transfers between GridFTP servers. A client is provided that is hosted in a Grid service, so it can be managed using the soft state model. The state of the transfer is stored in the service data and can be queried using the standard OGSI interfaces. It adds reliable and recoverable version of the Globus Toolkit globus-url-copy tool. That is, in case of a failure during the transfer, it is continued at a later time until it succeeds or a predefined termination condition is reached. Hence, problems such

as dropped connections, machine reboots and temporary network outages are dealt with automatically. RFT is heavily based on functionality provided by the Java CoG Kit GridFTP libraries, which are used to conduct the actual file transfer.

A persistent state of the service is maintained in a database. The RFT component uses the database to store the state of the transfer required for a restart after a failure. At present the RFT component uses PostgreSQL to store the state of the transfer to allow for the necessary information for restarts after failures. During the past two years various prototypes of RFT have been implemented (Madduri, Hood & Allcock 2002).

4.3. Swing-Based Java Application

Designing uniform user interfaces to connect a variety of data sources provides the most high level transparent abstraction to file transfer. In modern operating systems such interfaces are implicitly included. Examples are the KDE and the Windows desktop, which include file explorers. Additionally, commodity applications such as SecureFTP and LeechFTP provide easy-to-use graphical interfaces to file transfers between the client and a remote server. These tools provide an intuitive mechanism for managing and monitoring file transfers between clients and servers. We have designed such a tool and reported on its design in (von Laszewski, Alunkal, Gawor, Madhuri, Plaszczak & Sun 2003). We pose a number of simple requirements for the development.

Generality, Expandability, and Adaptability. It is advantageous to develop a sophisticated graphical user interface for file transfers in the Grid hiding the protocols from the user as much as possible. Hence, adaptations to future Grid protocols can be conducted easily. Such a graphical user interface would provide the end user with an abstraction that focuses on the pattern of the transfer instead of the details related to the transfer protocols.

Usability and Convenience. End users will require ease of use. The easiest way to achieve this is to mimic common tools that are already available for the desktop. Hence we promote the support of drag-and-drop interfaces and management tools for multiple file downloads and uploads, recursive directory transfers, and transmission management through queues. As such requirements are similar to Grid job submissions, the same graphical components ought to be reused as part of job management strategies.

Portability. The component to be developed ought to be portable. Although our requirements are language- and framework-neutral, we have implemented the

component in Java, which allows us to fulfill the requirements of easy porting and deployment as discussed in (von Laszewski et al. 2002). Hence, we have chosen the Java Swing framework to guarantee reuse with standard Java implementations.

Reusability and Composability. As the many of the tasks to manage file transfers are general, the underlying mechanisms to implement them ought to be exposed through a portable framework that encourages reuse. This can be achieved while abstracting the functionality and provide Java interfaces to them. Additionally, while embedding them in the Java Beans framework, we are able to utilize Java IDEs to assemble them. Exposure through a service model of the functionality further enhances the reusability aspect. However we clearly require these services to be able to be hosted on the client or the server side. Hence its deployment requirements must be minimal or adapted appropriately to the deployment resource.

Java Interfaces Abstracting Basic Functionality. While implementing our component we originally planned to develop service providers for an underlying protocol based on the Java Naming and Directory Interface (JNDI). However, after several failed attempts by a number of students to provide such JNDI-based providers, we decided to simplify our architecture further and develop a number of extremely easy to use interfaces, fulfilling a similar but more limited purpose. These interfaces are depicted in part in Figures 5 and 6.

```
interface Access {
  ...
  public void chdir(String name);
  public void list();
  public void mkdir(String name);
  public void rmdir(String name);
  public void rmfile(String name);
  public void rename(String from,
                     String to);
  ...
}
```

Figure 5: Accessing local and remote files and directories are elementary functions.

```
interface Transfer {
  ...
  public setFromUrl(String url);
  public setToUrl(String url);
  public startTransfer();
  public suspendTransfer();
  public resumeTransfer();
  public cancelTransfer();

  ...
}
```

Figure 6: Transferring files between locations is an elementary function.

Based on these elementary interfaces we have developed implementations using the FTP and GridFTP protocols. The graphical components that were developed

are based on these interfaces. Hence, adaptation to other protocols and file servers should be straightforward. The separation of concerns between access and transfer of files, as well as directories, allows the adaptation to be better abstractions than provided by JNDI, which provides predominantly solutions, to naming, binding, and searches.

A transfer service is implemented to manage transfers that are created through the graphical user interface events. The data movement is provided by using the Java CoG Kit `UrlCopy` which itself supports implicitly a number of protocols as discussed previously. Additionally, an interface to an early prototype of the RFT service that is described in more detail in Section 4.2 has been developed. Most of the advanced functionality of this component is integrated within the graphical user interface and as it is based on the Java interfaces depicted in Figures 5 and 6, where a URL is defined by `[protocol]://[user]@[host]:[port]:[file]`.

Graphical User Interface. Because the current generation of Web browsers does not implement in a sufficient way freely positionable, internal windows and the drag-and-drop paradigm, we implemented the component in Swing. A screenshot of the component is shown in Figure 7. Our prototype consists of file browsing and file transfer monitoring components. In part, these components are also implemented as Java Beans (*Java Beans* n.d.). Hence, they can also be integrated in a commodity interface development environment (IDE), such as JBuilder. This will bring us one step closer to a more convenient development environment for Grids. A screenshot of reusing the Beans in an IDE is shown in Figure 8.

The file browsing panel offers elaborate tree-based directory browsing and directory manipulation functionality. As a result, the user is able to (a) view the directory entries in tree structure, (b) transfer files with few clicks of the mouse, (c) create new directories, (d) copy files and directories, (e) delete files and directories, and (f) rename files and directories. The monitoring panel supports queuing of tasks representing file and directory transfers, checking of the status of these tasks, and managing transfers. Users have the flexibility to login to any number of FTP or GridFTP servers. Files in directories are displayed in a tree pane as part of internal frames. The user can perform basic file system operations, such as renaming, copying, and deletion. Additionally, the user can transfer files and directories by dragging and dropping them between the frames while observing the status in a monitoring window.

Since we have developed the component in Swing, it can be deployed through the Java Web Start technology. It can be used when the user has the ability to download and install files in his user space on the client computer. In essence it is not different

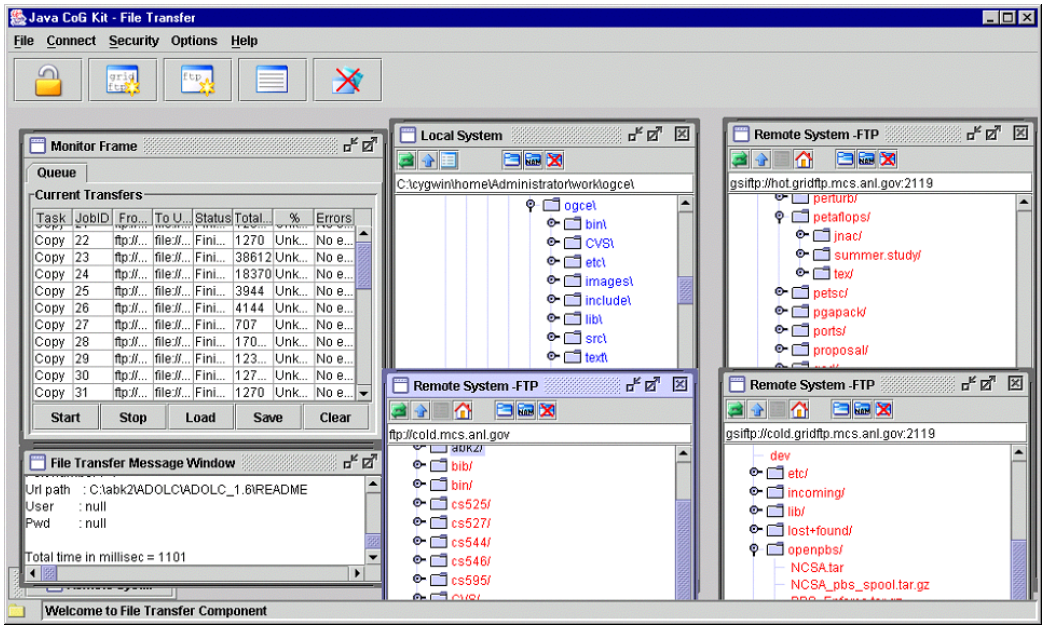

Figure 7: The GUI of the file transfer component.

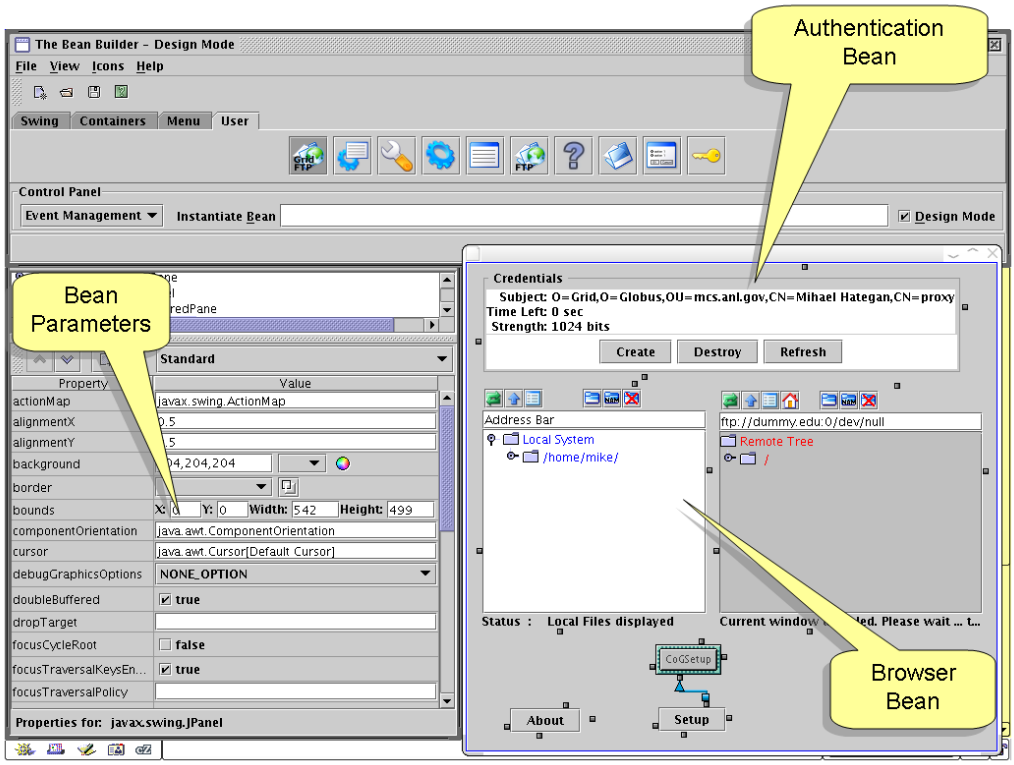

Figure 8: The GUI of the file transfer component in an IDE.

from installing a browser plug-in such as Acrobat Reader, a Microsoft program, or an MP3 player. Automatic updates provide an additional benefit while simplifying the maintenance and deployment of up-to-date clients. More information about the use of Web Start within Grids can be found in (von Laszewski et al. 2002).

4.4. Java Application With Plug-in Support

The Java CoG Kit project is currently prototyping a system called Entrada, which provides a GUI pane-based interface. But more importantly, it provides the ability to easily inform and supply the end user with plug-in upgrades. The framework is built on top of the jedit plug-in feature.

We have developed a GridFTP plugin is a Java Swing application using the Java CoG kit. It is a third-party transfer application with the ability to manage several transfers at one time. It displays these transfers in a tabular format referred to as the *transfer table*. The transfer table allows the user the easily manage transfers and monitor their progress. Other plug-ins in the Entrada framework can *hand-off* transfers to the GridFTP plug-in so that they can specialize in another task; and as such the sole task of the GridFTP plug-in is to manage these transfers.

The transfer table in the GridFTP plug-in consists of rows and columns. Each row represents a third-party transfer between two remote hosts. Each column displays a characteristic of that particular transfer (Figure 9).

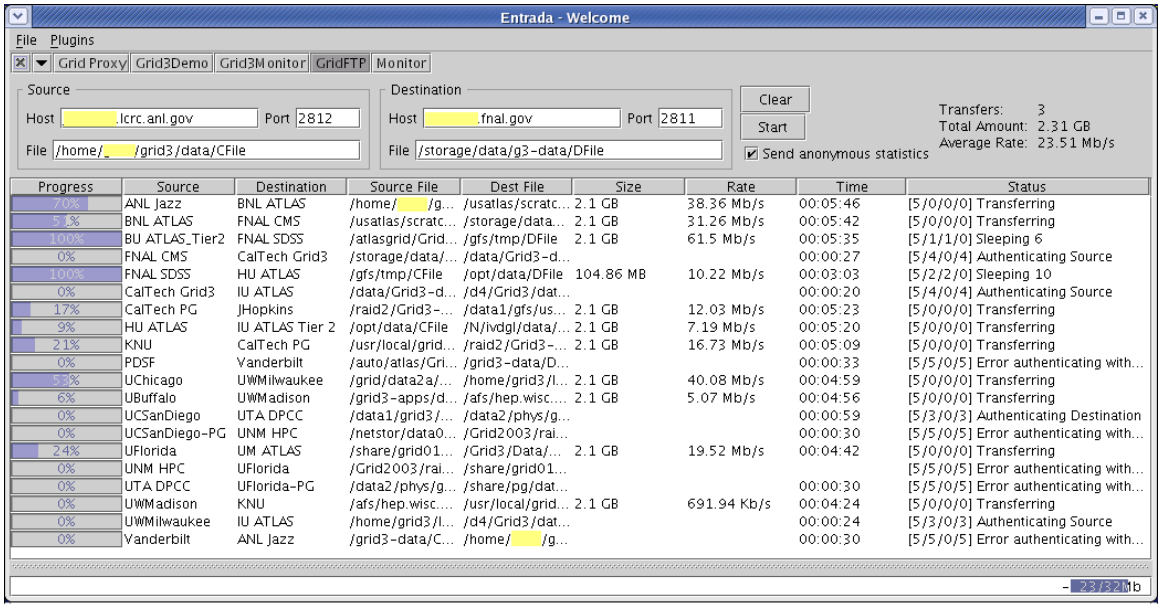

Figure 9: The GUI of the file transfer component.

The GridFTP plug-in is part of the Entrada framework, which enables other plugins the ability to easily use its *expertise* in transferring files. However, it is also built such that it can run as a stand-alone application if the user so chooses.

The GridFTP plug-in also has the ability to collect anonymous statistics on usage. Every time a successful transfer finishes, it contacts a server and records the size of the file transferred, the wall-clock time of the transfer, and the transfer rate. These statistics are collected with the goal of improving the plug-in through usage patterns visible in the data. Statistics gathering is completely optional and can be turned off by the client. If a GridFTP plug-in upgrade is release, all clients will see the upgrade available and can click on a checkbox to receive the upgrade. No hunting for packages, comparing version numbers, or reading documentation is required by the user.

4.5. GridAnt and Java CoG Kit Workflows

Recently we showed that through the proper abstractions we can design a framework that is independent from the underlying version of the Globus Toolkit as long as we focus on elementary patterns such as file transfer. In (Amin, Hategan, von Laszewski & Zaluzec 2004) we enhanced the Java CoG Kit significantly to include even more useful APIs for the developer community. At the same time we exposed such abstractions also as part of a sophisticated XML specification that is based on *ant*. Not only can we formulate simple workflows with our GridAnt in XML, but we can also display them in a sophisticated Grid Workflow viewer. Upon execution the workflow gets augmented during runtime with the state of the calculation. If an error occurs, the user can correct the error and continue with the rest of the calculation. Figures 10 and 11 show a screenshot and an XML specification of different applications using our GridAnt Framework.

5. EXPERIMENTAL RESULTS

One of our goals is to evaluate the ease of use of our libraries and its impact on commodity deployment infrastructure. That is, one downloads the toolkit installs it with the default parameters and continues the operation. This is in contrast to many other typical performance measurements, that are based on a great deal of performance tuning. However, as such tuning can not be conducted by non-IT knowledgeable scientists we need either tools that install for many cases nicely with apropiate performance characteristics preinstalled, or that tune themselves during use. Such deployment issues are often ignored in performance measurements. Hence we have decided to try an install without any tuning as we would expect by a non-IT user.

In our test we measure the time taken to conduct file transfers through various methods, including operating system methods, Globus C client programs, and Java CoG Kit client programs and components, and then to compare them with each other. We choose to run the tests using rcp, scp, and Globus C file transfer methods to define

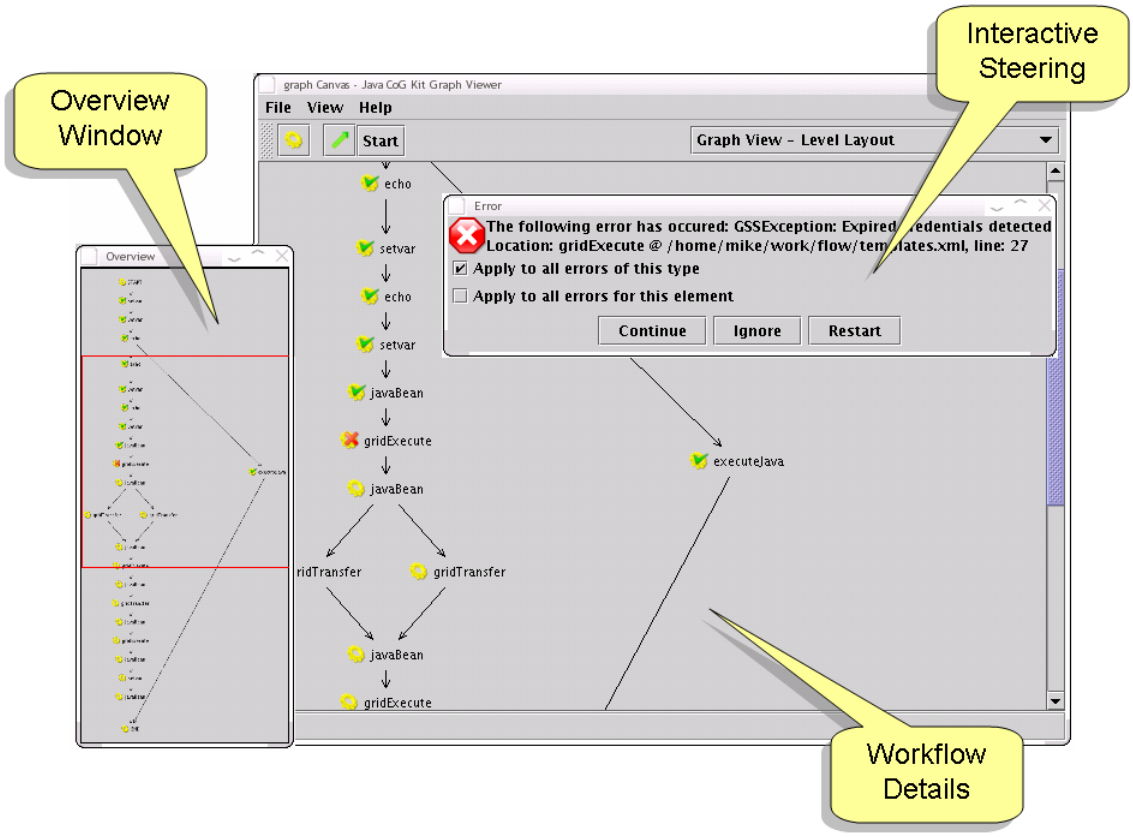

Figure 10: The GUI of the file transfer component in an workflow.

the baseline for comparisons, against the Java CoG Kit file transfer mechanisms. We also test the third-party file transfer capabilities provided by the Globus Toolkit version 3.0 (GT3), for additional comparison.

We conducted a number of elementary tests within a production environment consisting of two Pentium III 900 MHz dual-processor stations each having 512 MB of memory. The tests were run from partitions residing on a Seagate ST318451LC with a nominal transfer rate of 40 MB/s (st3 n.d.). The disks were driven by an Adaptec 7892P controller. Each station was equipped with a Syskonnect Gigabit Ethernet network card. Both stations resided on a switched 1 Gbit Ethernet local network. The Linux operating system (2.4.18) was running on both the stations. The Java Virtual Machine version 1.4.1 was used as the Java environment. The server ran the 2.4.0 version of the Globus Toolkit. On the client side, the Java CoG Kit version 1.1alpha was used.

Testing was performed under minimal network and CPU load conditions. An analysis of the usage patterns for the stations was made across a time-span of one week. Observations showed a decrease in CPU utilization and network load between 7 pm and 7 am, time during which the load conditions remained constant. All of the

```
<target name="sampleWorkflow">
    <sequential>
        <grid-setup/>
        <grid-authenticate/>
        <grid-copy name="copyInputFile"
                   provider="GT2"
                   security="xmlSignature"
                   delegation="limited"
                   from="gridftp://server1.ncar.edu/inputFile"
                   to="gridftp://server2.mcs.anl.gov/inputFile"
                   parallelStreams="4"
                   tcpBuffer="16384"/>
        <grid-execute name="climate"
                   provider="GT2"
                   server = "server2.mcs.anl.gov:1234"
                   security="xmlEncryption"
                   delegation="full"
                   executable = "mm5"
                   arguments="-file inputFile"
                   directory="/home/vonlaszewski"
                   localExecutable="false"
                   redirect="false"
                   outputFile="stdout.txt"
                   errorFile="error.txt"/>
        <grid-copy name="copyOutputFile"
                   provider="GT2"
                   security="xmlSignature"
                   delegation="limited"
                   from="gridftp://server2.ncar.edu/outputFile"
                   to="gridftp://laptop.mcs.anl.gov/outputFile"
                   parallelStreams="4"
                   tcpBuffer="16384"/>
    </sequential>
</target>
```

Figure 11: GridAnt XML specification.

tests were executed between 7 pm and 3 am. Repeating the tests on different days, during the same off-peak hours, showed less than 5 percent difference in the mean values.

The performance tests for file transfer were performed by successively transferring files of sizes small to moderate size, from one station to the other, using the various methods chosen. The mean transfer speed for each of the experiments is shown in Table 5. Charts showing the graphical representation of the data obtained for file transfer time, and transfer speed are shown in Figure 12 and Figure 13, respectively.

Table 2: File transfer mean times

Transfer Method	Mean Transfer Time for Different File Sizes in Seconds					
	1 MB	32 MB	64 MB	128 MB	256 MB	512 MB
C GridFTP	606.74	1869.80	3233.73	6119.71	21630.97	44139.68
Java GridFTP	1211.55	2129.36	3104.41	5238.17	11600.57	20929.86
RCP	456.07	945.89	1357.35	2345.30	7047.40	24363.46
SCP	961.57	4090.47	6629.99	11668.76	26722.91	58006.22

Our measurements show that using the Java GridFTP client is slower, compared with the C GridFTP client, for smaller files in the range of 1 MB to 32 MB. The difference being in the 3 to 5 second range, which can be attributed to the JVM startup costs. For larger file sizes, from 64 MB to 512 MB, the Java GridFTP client outperforms the C GridFTP client, taking only half the transfer time for file sizes from 256 MB to 512 MB. In general, any Grid application requiring large sets of data, and running in a similar environment, will benefit from using the Java CoG Kit on the client side. At the same time, more computationally oriented applications will incur a performance penalty that will be very small compared with the total execution time of such an application.

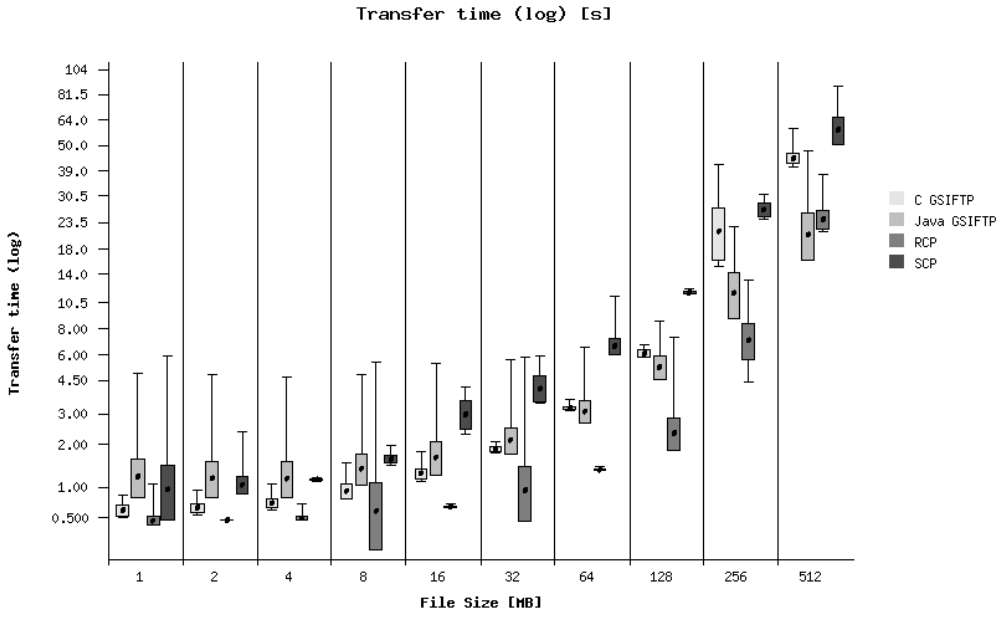

Figure 12: File transfer timing results in seconds.

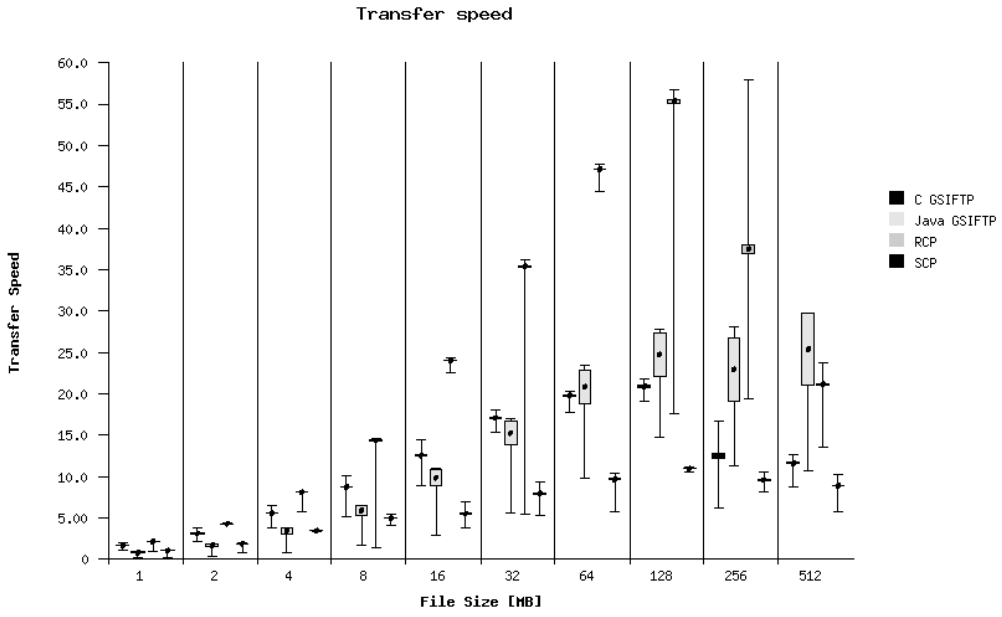

Figure 13: File transfer speed in MB/seconds.

6. CONCLUSION

We have developed a simple set of use patterns for file transfers in Grids. We have implemented them and delivered a Java library as part of the Java CoG Kit. Several sophisticated applications exist that use this library. Together with the community, we are working toward building user-friendly environments for interactive and transparent file access to Grid users. Such an interactive model is required for assisting scientists in their quest of accessing the Grid in a way that hides much of its complexity of the Grid and involves a large degree of interaction with the system. The Globus Toolkit 3 depends on parts of the Java CoG Kit and its file transfer components and APIs; hence it is distributed also in part with GT3. The Swing-based file transfer prototype component is not distributed as part of the Java CoG Kit but is available as add-on. We expect that with community efforts more components will be integrated into the Java CoG Kit distribution.

The Java CoG Kit clients offer the benefit of portability across platforms supporting the Java Runtime Environment. In the context of file transfer, the obtained data shows increased performance when using the Java CoG Kit for files larger than 64 MB for an untuned installation; in order to achieve high performance GridFTP servers and clients must be adjusted by an expert.

ACKNOWLEDGMENTS

This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38. Globus Toolkit research and development have been supported by DARPA, DOE, and NSF. This work would not been possible without the help of the Globus Project team. We thank all the members of the Globus Project for their valuable help. Globus Toolkit and Globus Project are trademarks held by the University of Chicago. We acknowledge Beulah Alunkal, who for the past two years worked on the development of a graphical user interface to Grid file transfers; this work resulted in several prototypes, on of which is described in Section 4.3. We also thank Shashank Shankar. We thank Dennis Gannon, Marlon Pierce, and the Indiana Extreme Team, and especially Ian Foster for their valuable discussions and support. The Java CoG Kit is supported by NSF and DOE.

REFERENCES

1. Allcock, W., Bester, J., Bresnahan, J., Meder, S. & Tuecke, S. (2003). GridFTP: Protocol Extensions to FTP for the Grid, GWD-R. `http://forge.gridforum.org/projects/ggf-editor/document/GFD.20/en/1`

2. Amin, K., Hategan, M., von Laszewski, G. & Zaluzec, N. J. (2004). Abstracting the grid, *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2004)*, A Coruña, Spain.

3. Berman, F., Fox, G. C. & Hey, T. (eds) (2003). *Grid Computing: Making The Global Infrastructure a Reality*, Wiley.

4. Debis, J. (n.d.). LeechFTP, `http://www.iweb.net.au/Downloads/leech.html`.

5. Foster, I. & Kesselman, C. (eds) (2003). *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers.

6. Foster, I., Kesselman, C., Nick, J. M. & Tuecke, S. (2003). *Grid Computing: Making the Global Infrastructure a Reality*, Wiley, chapter The Physiology of the Grid, pp. 217–249. `http://www.globus.org/ogsa`

7. Foster, I., Kesselman, C., Tsudik, G. & Tuecke, S. (1998). A Security Architecture for Computational Grids, *5th ACM Conference on Computer and Communications Security*, ACM Press, pp. 83–92. `ftp://ftp.globus.org/pub/globus/papers/security.pdf`

8. Gannon, D., Fox, G., Pierce, M., Plale, B., von Laszewski, G., Severance, C., Hardin, J., Alameda, J., Thomas, M. & Boisseau, J. (2003). Grid portals: A scientist's access point for grid services, *Ggf working draft*, Global Grid Forum GCE-WG. (Draft 1). http://forge.gridforum.org/projects/ggf-editor/document/GCE-Portal-working-draft/en/1/GCE-Portal-working-draft.pdf

9. *Java Beans* (n.d.). Web Page. `http://java.sun.com/products/javabeans/`

10. *Java CoG Kit* (n.d.). Web Page. `http://www.globus.org/cog/`

11. Madduri, R. K., Hood, C. S. & Allcock, W. E. (2002). Reliable file transfer in grid environments, *27th Annual IEEE Conference on Local Computer Networks*, IEEE Computer Society, Tampa, FL, pp. 737–738.
http://computer.org/proceedings/lcn/1591/15910737.pdf

12. Novotny, J. (2003). *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley, chapter The Grid Portal Development Kit.
http://dast.nlanr.net/Features/GridPortal/

13. Plaszczak, P., Link, J., Wellner, R. & Hubbard, P. (n.d.). GridFTP 1.0 Explained.
http://www-unix.mcs.anl.gov/~pawel/ftp/GridFTPexplained.doc

14. Postel, J. & Reynolds, J. (n.d.). File Transfer Protocol, RFC.
http://www.w3.org/Protocols/rfc959/Overview.html

15. *SecureFTP* (n.d.). Web Page.
http://www.glub.com/products/secureftp/

16. st3 (n.d.). Seagate ST318451LC Configuration and Specifications.
http://www.seagate.com/support/disc/specs/scsi/st318451lc.html

17. von Laszewski, G. (1996). An Interactive Parallel Programming Environment Applied in Atmospheric Science, *in* G.-R. Hoffman & N. Kreitz (eds), *Making Its Mark, Proceedings of the 6th Workshop on the Use of Parallel Processors in Meteorology*, European Centre for Medium Weather Forecast, World Scientific, Reading, UK, pp. 311–325.
http://www.mcs.anl.gov/ gregor/papers/
vonLaszewski–ecwmf-interactive.pdf

18. von Laszewski, G., Alunkal, B., Gawor, J., Madhuri, R., Plaszczak, P. & Sun, X.-H. (2003). A File Transfer Component for Grids, *in* H. Arabnia & Y. Mun (eds), *Proceedings of the International Conferenece on Parallel and Distributed Processing Techniques and Applications*, Vol. 1, CSREA Press, Las Vegas, pp. 24–30.
http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gridftp.pdf

19. von Laszewski, G. & Amin, K. (2004). *Grid Middleware*, Wiley, chapter Middleware for Commnications. to be published.
http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-gridmiddleware.pdf

20. von Laszewski, G., Blau, E., Bletzinger, M., Gawor, J., Lane, P., Martin, S. & Russell, M. (2002). Software, Component, and Service Deployment in Computational Grids, *in* J. Bishop (ed.), *IFIP/ACM Working Conference on Component Deployment*, Vol. 2370 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 244–256.
`http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--deploy-32.pdf`

21. von Laszewski, G., Foster, I., Gawor, J. & Lane, P. (2001). A Java Commodity Grid Kit, *Concurrency and Computation: Practice and Experience* **13**(8-9): 643–662.
`http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--cog-cpe-final.pdf`

22. von Laszewski, G. & Wagstrom, P. (2004). *Tools and Environments for Parallel and Distributed Computing*, Series on Parallel and Distributed Computing, Wiley, chapter Gestalt of the Grid, pp. 149–187.
`http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gestalt.pdf`