Storage Manager and File Transfer Web Services*

William A. Watson III#, Ying Chen, Jie Chen, Walt Akers

Thomas Jefferson National Accelerator Facility

Newport News, Virginia 23606, U.S.A.

Abstract

Web services are emerging as an interesting mechanism for a wide range of grid services, particularly those focused upon information services and control. When coupled with efficient data transfer services, they provide a powerful mechanism for building a flexible, open, extensible data grid for science applications. In this paper we present our prototype work on a Java Storage Resource Manager (JSRM) web service and a Java Reliable File Transfer (JRFT) web service. A java client (Grid File Manager) on top of JSRM and JRFT is developed to demonstrate the capabilities of these web services. The purpose of this work is to show the extent to which SOAP based web services are an appropriate direction for building a grid-wide data management system, and eventually grid-based portals.

KEY WORDS: web services, XML, grid, data grid, meta-center, portal

1. Introduction

Built upon a foundation of SOAP, WSDL and UDDI technologies, web services have become a widely accepted industry standard in the last few years [1] [2]. Due to their platform independence, universal compatibility, and network accessibility, web services will be at the heart of the next generation of distributed systems. As more vendors offer SOAP tools and services, the advantages of using SOAP and web services as an integration point will become even more pronounced.

The grid computing community has also recognized the importance of using web services. The Globus project has proposed adding a web service layer to its existing infrastructure, and has proposed an Open Grid Services Architecture (OGSA) [3]. The Unicore project has already developed an OGSA-compliant web services wrapper for their grid toolkit [4]. Other forums are likewise engaged in developing web services for grids and portals [5].

In the Fall of 2000, ahead of the major projects' move to web services, Jefferson Lab started to investigate the feasibility of providing grid capabilities using XML based web services. These XML services were later migrated to SOAP based web services. The goal of this ongoing work is to present all available resources in the grid to the users as a single virtual system, a computing meta-facility. To achieve this, many loosely coupled web systems, such as a distributed data grid, a distributed batch system, etc. are required. In this paper we will emphasize file management services, focusing upon two of the services and summarizing several others.

The requirements of the Jefferson Lab distributed data analysis are described in a previous paper [6]. In that paper, the emphasis was on the laboratory's existing computing infrastructure, and a web services layer architecture. This paper will concentrate more on the implementation details and the lessons learned from the first prototypes of the data management web services.

1.1 Motivation

The web has been overwhelmingly successful at providing seamless access to a wide range of information, and at providing a basic level of interaction with distributed systems (for example, electronic commerce). Web services seeks to exploit this successful distributed architecture by adding a messaging layer (SOAP) and

---

* Work supported by the Department of Energy, contract DE-AC05-84ER40150.

Correspondence to: William Watson, Jefferson Laboratory MS 16A, 12000 Jefferson Av, Newport News, VA 23606. Email: Chip.Watson@jlab.org

a syntax for constructing messages (XML, WSDL) on top of the web protocols. This allows one to exploit all of the existing features such as secure http to build a loosely coupled, distributed system.

As an example of this leveraging, it is very simple for a user with valid X.509 credentials to use resources across the world without having to log into individual systems, because https defines a mechanism for using SSL to connect to a server using these credentials. Using SOAP over https, one can submit a batch request to a meta-facility which automatically and optimally assigns the job to a particular compute site. This activity would include staging input files, running the job, and then automatically archiving the outputs from the job. Just as the web is easily extended by adding a new web server, it will be easy to add a new compute or storage node to this web based meta-facility.

1.2 Architecture

The web services grid we envisage contains many components arranged in three layers (Figure 1). The lowest level contains site-specific back-end services to manage disks and tertiary storage (a silo) and to schedule batch jobs onto local compute nodes. The middle layer provides a standard interface to these resources. It may be a thin layer (protocol translation and standardization) or may contain considerable business or management logic (as in our implementation). On top of these layers are the user interfaces and other client applications. The top layer needs no knowledge of the often complicated and site-specific lowest level (abstraction), so our design goal is to hide this lower layer and only expose an easy-to-use web services middle layer to all top-level applications.

The only exception to this principle is bulk data transfer, where the application can directly contact a file transfer daemon. This multi-protocol approach follows the web in general where there exist many file transfer and streaming protocols along side http.

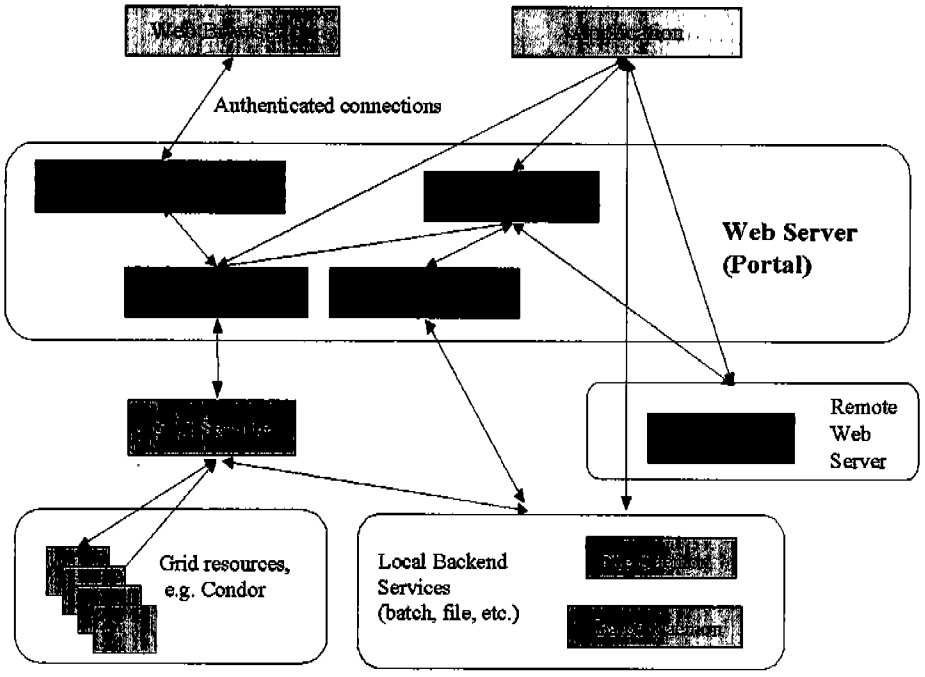

Figure 1: A Three Tier Web Services Architecture

The foundation of web services is SOAP – XML messaging over standard web protocols such as http. This lightweight communication mechanism ensures that any programming language, middleware, or platform can be easily integrated. Using SOAP, exchanging structured information in a decentralized, distributed environment is straightforward. SOAP mandates an XML vocabulary that is used for representing method parameters, return values, and exceptions. Although SOAP is not yet an Internet standard, the W3C Working

Draft does provide a way to explore the protocol and decide whether it is appropriate to solve the task of deploying a meta-facility.

2. Data Grid Web Services

2.1 Data Grids

Jefferson Lab is part of a collaboration funded by the Department of Energy's Scientific Discovery through Advanced Computing (SciDAC) program. The Particle Physics Data Grid Collaboratory [7] seeks to exploit grid technologies within high energy and nuclear physics experiments world-wide. PPDG is working closely with counterparts in Europe to address the needs of the Large Hadron Collider now under construction at CERN. In the short run, PPDG is using current large experiments as testbeds for production data grids. Within this context, Jefferson Lab is deploying a testbed for data grids based upon web services.

One activity within PPDG has been to produce the functional specification of a storage management system or SRM. This specification [8] was developed collaboratively by experts in storage systems at many of the major high energy and nuclear physics labs around the world. These labs are well knows to have the most demanding data storage requirements of any scientific discipline.

The defined SRM operations include moving files into or out of a tape silo or managed disk system, changing a file's eligibility for deletion (pin/unpin) and marking a file permanent (can't be deleted from disk until it exists on permanent storage), and reporting the file status.

These SRM services have now been implemented as SOAP web services, using Java Servlet technology. Here, we name our resource management web service JSRM, for **J**ava **S**torage **R**esource **M**anagement. JSRM implements a superset of the PPDG SRM functionality and provides services for both managed and unmanaged file systems (where by "unmanaged" we mean conventional user and group managed disk space). JSRM contains functionality for accessing unmanaged file systems, and can also serve as a front end or gateway to a another (simpler) SRM system to perform silo and disk pool operations (Jefferson Lab's JASMine storage management software in our case [9]). JSRM is thus a web entry point to all the resources of a single site, including a user's home directory. A collection of such sites (when linked with additional services) forms a data grid.

The majority of the service request/response data structures of JSRM are defined according to the SRM functional specification document. Additional operations not defined in the current SRM specification may be added in a future version.

The remaining discussion of functionality uses the following terms from the SRM spec:

- GFN — Global file name; the name by which a file is known across the grid. This file is typically the primary index in a catalog, and contains no location information
- SFN — Site file name; the name by which a file is known by a data grid node or site
- SURL — Site URL (Uniform Resource Locator), a concatenation of the SOAP endpoint used to communicate with a site and the SFN
- TURL — Transfer URL, a standard web URL which includes a protocol specification for transferring a file, a file server hostname, and a local file path; example: ftp://hpcfs1.jlab.org/some-path/some-filename

Generally, the SURL is persistent, but the TURL can change if the local site migrates a file from one server to another.

2.2 File / Dataset Catalogs

In the PPDG model, a GFN can either be known in advance, or can be discovered by consulting an *application meta-data catalog*, which allows mapping from domain specific meta-data (such as beam energy or target specification for Jefferson Lab) into a set of file names (GFNs). The set of GFN's forms a namespace similar to the namespace of a Unix file system (recursive directories, etc.).

For optimal performance in a wide-area distributed system, a particular dataset or file may exist at multiple locations on the data grid, and so a *replica catalog* is used to map between the global name and location information, in other words to convert each GFN into one or more SURLs. Jefferson Lab has implemented a

ReplicaCatalog web service to hold the GFN namespace, and to map between GFN and SURLs. This service is implemented as a combination of a java servlet and an SQL database (the persistence layer for the namespace), and is accessible both as a SOAP web service, and as browsable web pages (via style sheets).

For our replica catalog, we have also implemented the notion of soft links, which allows a user to create a directory holding (links to) a set of files located in various other directories within the replica catalog. Links can also point to directories, or even to other links.

Because the replica catalog holds a file-system-like namespace, many of the operations supported by this service are designed to be identical to the operations supported by the storage manager web service (described below). This set of common operations includes *list, mkdir, rmdir, delete, status*. Through this design, it is possible to treat the replica catalog and the storage manager web service polymorphically, as is done by the Grid File Manager application (described below).

We have not yet implemented an application meta-data catalog, but it, too, is likely to be implemented with an SQL database as the persistence layer. There may be advantages to co-locating the application meta-data catalog and the replica catalog so that the replica catalog becomes just a view of a larger set of data management information, and this will be explored in future work.

2.3 Storage Resource Management Web Service - JSRM

The first web service component we implemented is an interface to the software responsible for the site's storage resource management. At Jefferson Lab, the various experiments are able to produce up to one terabyte of data daily, and this data is held in a 12,000 slot StorageTek silo. In addition, a group of disk pools exists to store calibration and analysis data, and data files currently in use. These disk pools are managed by a daemon (a disk cache manager) with site-specific file deletion policies. We use the term "managed file system" to refer to these disk pools. Each lab typically develops its own software to manage all of its storage resources including the mass storage system and cache disk pools. Although these software packages may be written in different languages and with different designs, they are performing the same task - storage resource management (SRM).

The JSRM component provides the following major services:

1. Directory listings, including file meta-data (size, owner, cached, pinned, permanent, etc.).
2. Mapping from SFN to TURL to read/write files from/to this site, including protocol negotiation and space allocation. The implementation allows for plug-able protocols, and the current implementation supports http, ftp, and jparss (see below).
3. Translation between SFN or SURL and a local file path, allowing local applications to find a grid file on a locally mounted disk without needing to know any local mount point naming conventions.
4. File status change operations, such as stage a file, pin a file, migrate or copy a file to the silo.
5. File system management (copy, delete, make directory, etc., operations), including operations on managed and unmanaged areas.

SOAP is used as the service protocol: requests and responses are in XML, which is human readable and can be described by Web Services Description Language (WSDL). The WSDL for JSRM is available online [10].

Due to the platform independence and widely available tools and packages [11][12], Java has been selected as the implementation language. The architecture of JSRM is illustrated in Figure 2. The functionality is distributed among several java classes: one java servlet is responsible for getting the SOAP requests from the user, validating the SOAP message, and passing it to an appropriated server object. Site-specific information (such as the set of available file systems) is configured via an XML configuration file, which is loaded into a java object and passed to the individual function server by the java servlet. With this design, an installation can be configured differently according to local policy (such as allowing access to the user's home directory) and file storage structure.

SRM operations for unmanaged disk areas are handled directly by JSRM. This allows JSRM to be deployed at a site which has only user managed disk space (no disk management system, no tertiary storage system).

JSRM can also be layered above an existing storage manager. Certain operations on the managed disk areas and silo are then performed by deferring the requests to a separate object or SRM, which can itself be a web

service or a java class. As long as every site uses the same SRM or java interface, JSRM can be deployed as a gateway, adding functionality without modifying the secondary SRM's code.

The Java API for XML Messaging (JAXM) is used to build, send, receive, and decompose the SOAP messages in this work. The current version of JAXM implements SOAP 1.1 with Attachments messaging. It takes care of all low-level XML communications – JSRM servers focus only on providing the service the user has requested.

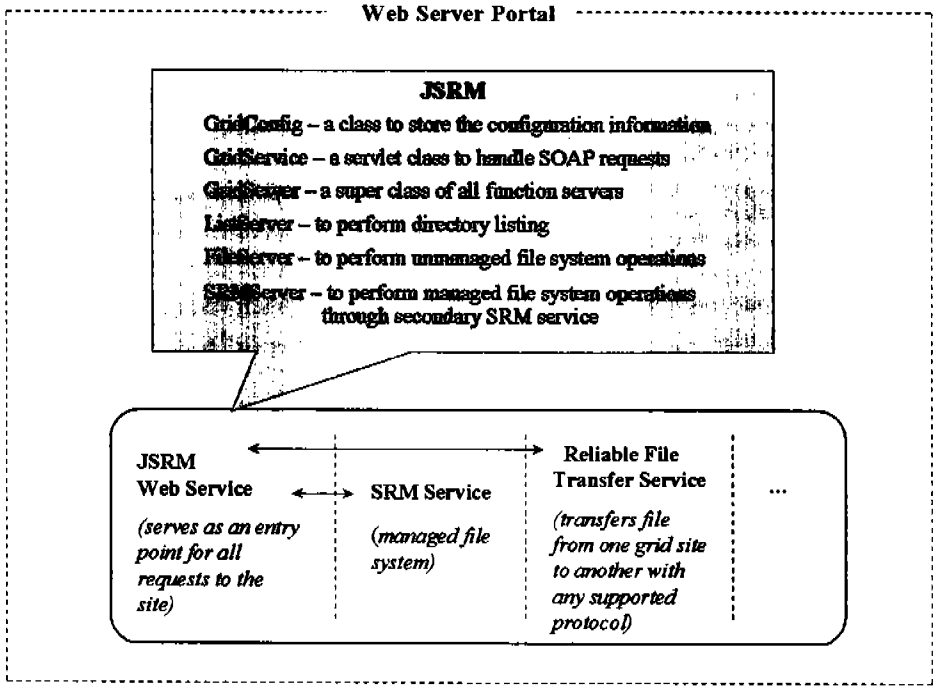

Figure 2: The Architecture of the JSRM Web Service

2.4 Reliable File Transfer

The next essential component for a data grid is a service to move datasets from one grid node (site) to another, otherwise known as third party file transfers. This service must be reliable, gracefully handling intermittent network or node unavailability problems.

A reliable file transfer service could be included in the site JSRM service (additional methods of the same web service) However, in this project, we decided to implement it as an independent java server class with a web service interface. In this way, Reliable File Transfer functionality can be easily invoked by JSRM or serve as a user callable web service. This demonstrates the power and simplicity of web services: any web service written in any language can interact with any other web service.

To ensure the reliability of the transfer request, a MYSQL database is used to store every valid request. The request status, any error which occurred during the transfer, the time stamp of each step, and other relevant data are saved in a database table. Once an operation completes, the summary history of the transfer is also kept in the database for statistical use in the future. The architecture of the Reliable File Transfer web service is illustrated in Figure 3.

When the JRFT server receives a valid SOAP request, it first adds the request to the queue, and then returns a response to the user. This response only contains one element – transfer request-id – and with this the user can later check the transfer status. The decision to implement a pull architecture for such long-lived transactions is intentional – it allows the initiating client to exit, and then come back later, perhaps long after the transfer has completed, to check for errors. This loosely coupled style simplifies system construction and improves

flexibility, with only a minor increase in overhead to poll (which in any case is vanishingly small compared to the cost of moving large datasets).

Once a request is queued, there is a background thread running constantly checking the queue to find files to transfer. For each request, it must first negotiate the transfer protocol with the source site's JSRM and destination site's JSRM. If there is an agreement between both sites, the static method getTransferClient (String protocol_name) in the TransferClientFactory class will be called to obtain the corresponding transfer client to perform the real file transfer.

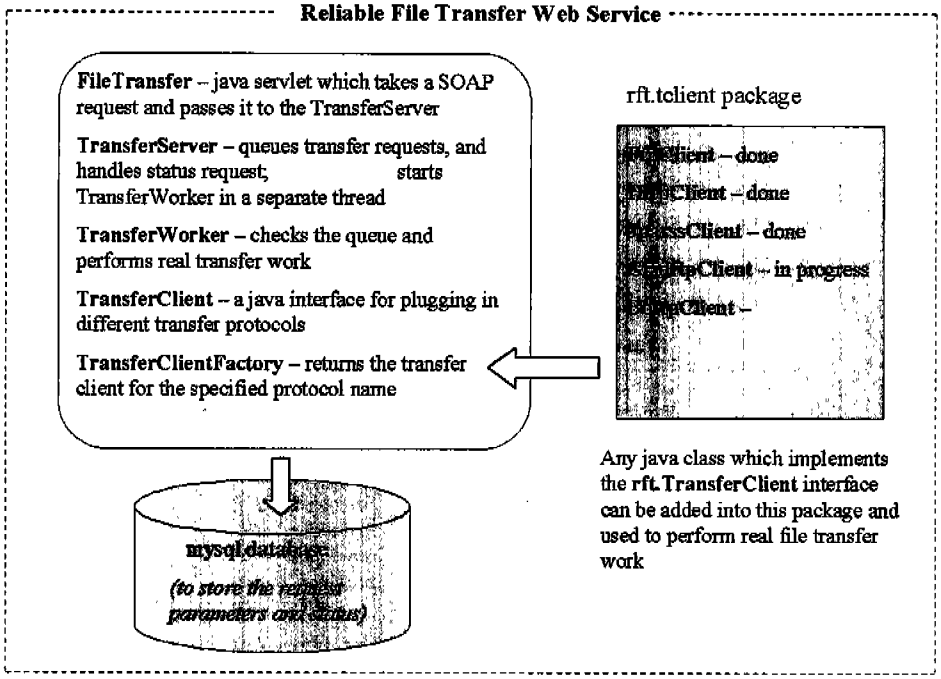

Figure 3: The Architecture of the JRFT Web Service

In order to discover the set of transfer clients (protocols) automatically, all available transfer clients must implement TransferClient interface and follow a certain naming convention, e.g. they must be named as ProtocolnameClient.java (only the first letter of the protocol name is capital and the remaining letters of the name are in lower case). So far there are three transfer protocol clients implemented in the rft.tclient package. FtpClient and HttpClient are used to transfer files between sites for which authentication is not needed. JparssClient uses the Java Parallel Secure Stream (JPARSS) package developed by Jefferson Lab [13] to perform authenticated parallel file transfers. Additional transfer clients using different protocols can be plugged in dynamically without any modification to the existing code (additional protocols will be added as time permits).

Depending upon the underlying file transfers capabilities, the RFT service might be at a different site from both the source and destination sites. For simple protocols like ftp and http, only pull is implemented, and so the destination must be the same as the RFT site. (Request forwarding from one RFT instance to another is foreseen to deal with these protocol restrictions). Each protocol's capabilities are described in XML, so the RFT service discovers these restrictions dynamically, and uses them in the protocol negotiation.

When a file transfer process has failed, SenderException or ReceiverException will be thrown by the TransferClient. SenderException is thrown only when a fatal error occurs, for example when the source SURL is invalid, or authentication has failed. When the transfer fails due to a possibly transient error, e.g. database error, network problem, etc., ReceiverException will be thrown and the request will be added back to the queue and will be tried again later.

After submitting a file transfer request, a user can check the transfer status. If the transfer has already started, additional information such as the number of bytes transferred and average bandwidth obtained will be included in the SOAP response.

2.5 Security

The security mode used for these web services is certificate-based browser-model security, which means all privileged services are provided only through https/ssl connections. A permanent X.509 user certificate or a temporary proxy certificate [14] is required to use these services. The server maps (using a map file) the subject of a user's certificate to the local user account to perform privileged operations using operating system access control.

In some cases, we have found it useful to allow some limited superuser operations on the grid, analogous to those performed by privileged daemons on a local system. As one example, the replica catalog allows entries to be made by certain privileged users (daemons using server certificates known to the replica catalog) to have the owner field set to a name other than the name of the calling user. In this way we have built automatic file registration "spiders" which do not need an individual user's proxy certificate to function. In each such case, the extended privileges just allow the special caller to do operations as if he were a different user.

3. Grid File Manager

3.1 Grid File Interface – a Java API

To make these file management web services easier to use, it has been found to be helpful to develop a client library to hide the somewhat complicated SOAP request and response operations from a typical user. A Grid File Interface has been designed to wrap the communication between the service providers into a simple java API (Fig. 4). A soap implementation of this Grid File Interface, based upon JSRM , JRFT, and the ReplicaCatalog has been done.

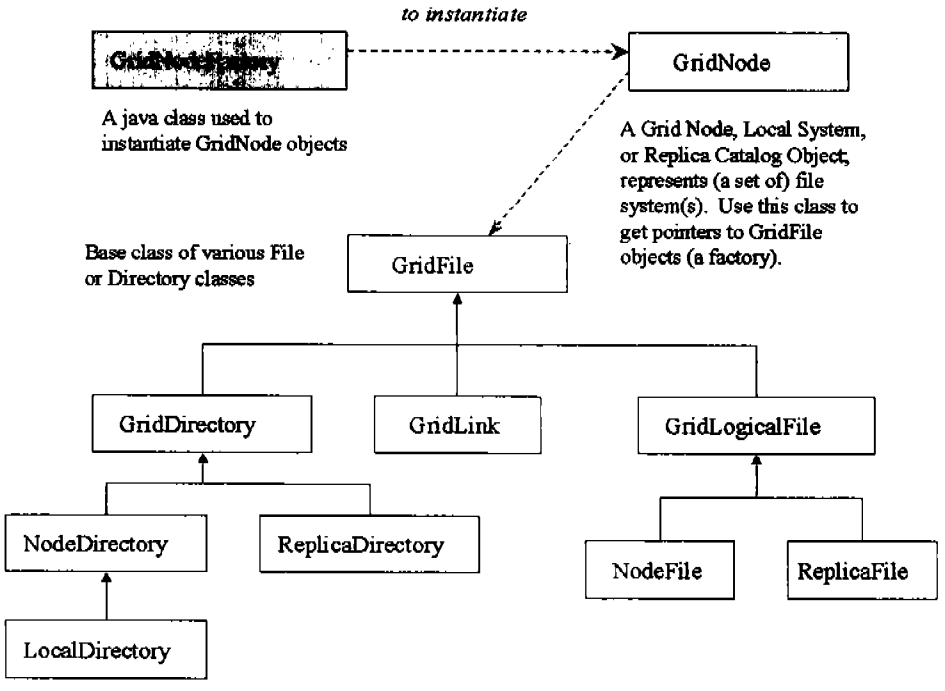

Figure 4: Grid File Interface Inheritance Tree

The GridNode object represents a grid node or physical site. It has a name, a URL, and other properties. A static method, getGridNode in GridNodeFactory will return a corresponding GridNode object with a specified URL. By using the list method in a GridNode class you can obtain a pointer to a given directory, file, or link.

GridDirectory class, like File in the java.io package, provides file system manipulation methods, such as delete, link, mkdir, addFile, list, etc. The client classes implemented with these interfaces then deal with the SOAP service request and response. In the addFile implementation, the Reliable File Transfer web service is used to perform the data movement when the source and destination are on different sites, and the corresponding JSRM service is used when a file copy within a site is requested. Using the classes defined in this interface, managing the resource on the grid is as easy as managing a local file system (as for local Java clients with the java.io.File class).

While the current implementation of this client library is done in Java, it would be straightforward to also produce a corresponding C++ client library, by building upon existing C++ SOAP implementations.

3.2 Grid File Manager

A Grid File Manager user interface is developed on top of the grid file API. This is a java application client and is used to manage the resources located on different systems within the grid, in the same or different organizations. An xml configuration file (accessed over the network via a standard URL) configures the systems that the Grid File Manager tool manages. In our version 1.0 release, the user is able to choose either his configuration file or the default one. Included in this configuration file are pointers to the replica catalog and a set of initial grid nodes, so that this application can be deployed without a web services directory service. At some point the URL to a configuration file could instead be a URL to a UDDI repository (directory).

Any server site that provides JSRM and JRFT services can be manipulated via the Grid File Manager. In addition, the tool can also manage the user's desktop computer (local system) without the need for locally installed web services.

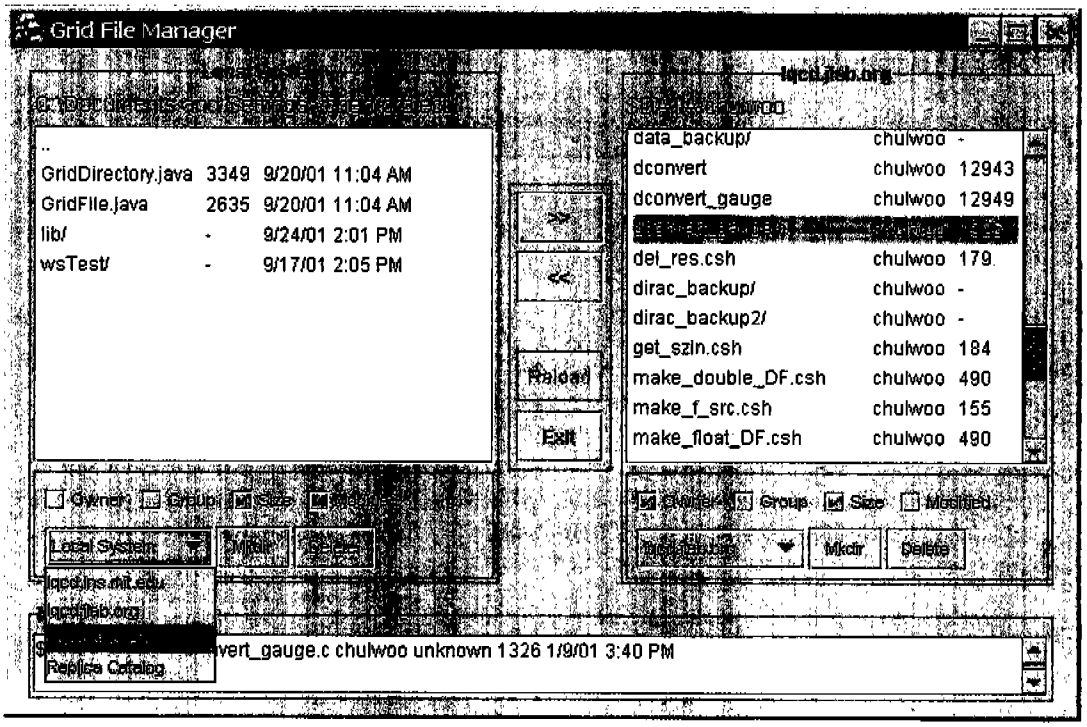

Fig. 5 Grid File Manage Interface

Grid File Manager provides a graphical interface for the user to request services. All underlying communications with service providers are hidden. With this tool, the user can browse and manage the supported file system within the grid, and his entire file system on the local computer, including deleting files and directories, making new directories, obtaining file or directory meta-data, etc. To transfer a file or an entire directory tree from one site to another is as easy as a single click.

There are two identical file browser panes in the Grid File Manager interface, as shown in Fig. 5. They serve as source and destination when you request a file copy/transfer.

In addition to the grid node and local system, Grid File Manager can also browse the Jefferson Lab Replica Catalog system [15], but manipulations of the replica catalog other than "list" are not implemented in current release. It is foreseen that "drag-N-drop" operations from the ReplicaCatalog to a grid node or local system will imply a hidden look-up operation on the file followed by a transfer from a site holding the file, and operations "copying" a file to the ReplicaCatalog will imply registering the file in the catalog. These operations will be added in a future release.

We use Java Web Start [16], an application deployment technology from SUN, to launch and manage the resources of Grid File Manager. By clicking on a web page link, Java Web Start automatically downloads all necessary application files to the user's computer if the application is not present. These files are cached on the user's computer so it is always ready to be re-launched anytime, either from an icon on the desktop or from the browser link. Each time the application is launched, Java Web Start will check the server for a new version of the application, and automatically download it if a new version is available. If not yet installed by the user, the user will be prompted to download the Java Web Start tool and browser plug-in. This technology ensures that users are always using the latest, correct version of software, and eases the deployment workload.

The security model used for this application is X.509 credential based, with short lived credentials (proxies) been given to a server to allow it to act on client's behalf. For authentication purposes, a Jefferson Lab or DOE Science Grid certificate is needed to use the Grid File Manager against the Jefferson Lab nodes. When launching an application, the user will be prompted for the pass phrase of his local certificate repository if no valid proxy exists, and a 24-hour life proxy certificate will be generated. This proxy will be used to connect to the server for privileged service operations. Within the 24-hour period, the user can use the application without entering a pass phrase again.

4. The Lattice Portal

The components described above are currently being deployed onto a testbed consisting of Jefferson Lab and the MIT Lab for Nuclear Science, for use by the Lattice Hadron Physics Collaboration. This small not-quite-meta-facility includes a cluster of 48 processors at MIT (alphas), a small cluster of 40 alphas at Jefferson Lab, and a cluster of 128 pentium 4 nodes at Jefferson Lab (to be upgraded in the Fall of 2002 to at least 256). The data grid tools are being used for moving lattice simulation files between the sites, and between sites and desktops. An additional node at the University of Maryland will be added Fall 2002, a possibly also a node at Oak Ridge National Lab in the same timeframe.

This testbed also serves to test software for use within a national Lattice QCD (quantum chromo-dynamics) collaboration also funded by the Department of Energy's SciDAC program. This larger collaboration plans to deploy a distributed 20+ teraflops (sustained) facility within 4 years with major installations at Jefferson Lab, Fermilab, and Brookhaven National Lab, and will use data grid technology to manage the resulting simulation datasets.

5. Lessons Learned and Future Plans

A well constructed set of web services, using SOAP as the protocol, have been shown to be useful building blocks for creating distributed systems with standard interfaces. The JSRM web service combined with Reliable File Transfer provides the most useful services to our cluster users. It provides an example of utilizing cutting-edge technologies to solve a key grid computing problem.

Within this data grid project, good object oriented design principles have allowed us to incorporate multiple file transfer protocols (via interface implementation), and have allowed us to do a large set of abstract file system operations on both the actual storage resource and the replica catalog polymorphically, easing the work of implementing the Grid File Manager.

As we have begun to deploy these tools into a production environment, the value of privileged processes on the grid has become clear, and such processes may provide an appropriate mechanism for dealing with the problem of expired user proxies associated with long running batch jobs or file transfer jobs.

In the next step of this prototyping work, we will investigate a different security mode, IETF XML signature (Digital Signature). Such signatures could allow for finer grained authorization of delegated work, and could help to limit the scope of privileged grid processes so as to prevent a compromise of the grid at one point from propagating to the entire grid.

So far, the JSRM implementation is being used primarily with unmanaged file systems, with limited interaction with the JASMine cache manager and silo through a java interface. JASMine is currently being modified to have an SRM interface (conforming to the PPDG spec), and when that is finished JSRM will be converted to forward managed file system operations to a secondary SRM web service, with JASMine as the first implementation to be tested.

For the Grid File Manager, we will add the remaining Replica Catalog functions: virtual read (lookup and fetch), write (publish, perhaps including a push to a grid node if the data source is the local desktop), and delete (unpublish). We are also producing a number of command line tools for fetching and publishing data sets.

References

1. Glass G, The Web Services (R)evolution: Part 1, http://www-106.ibm.com/developerworks/library/ws-peer1.html. [July 5, 2002].
2. Waters J., Web services: The Next Big Thing?, http://www.adtmag.com/article.asp?id=6124 [July 5, 2002].
3. See http://www.globus.org/research/papers/ogsa.pdf [July 5, 2002].
4. The Unicore Project, http://www.unicore.org/ [July 5, 2002]. The web services implementation can be found at http://www.unicore.org/downloads.htm looking in the index for OGSA.
5. See, for example, http://gridport.npaci.edu/pubs/workshops/gce/webservMay02/ [July 5, 2002].
6. Watson W, Bird I., Chen J., Hess B., Kowalski A, Chen Y, A Web Service Data Analysis Grid, *Concurrency Computat.: Pract. Exper.* 2002 (14).
7. Particle Physics Data Grid (PPDG), http://www.ppdg.net/ [July 5, 2002].
8. Shoshani A, Sim A, Gu J, Storage Resource Managers: Middleware Components for Grid Storage, *Proceedings of the 18th IEEE Symposium on Mass Storage Systems* 2001.
9. Building the Mass Storage System at Jefferson Lab, by Ian Bird, Bryan Hess, Andy Kowalski, *Proceedings of the 18th IEEE Symposium on Mass Storage Systems* 2001.
10. See http://lqcd.jlab.org/wsdl/jsrm.wsdl [July 5, 2002].
11. Apache Java project, http://java.apache.org/ [July 5, 2002].
12. Sun xml web site, http://java.sun.com/xml/ [July 5, 2002].
13. Chen J, Watson W, JPARSS: A Java Parallel Network Package for Grid Computing *Proceedings of the 2nd International Workshop on Multimedia and Intelligent Networks*, 2002, p944. March, 2002, Raleigh, NC.
14. Foster I, Kesselman C, Tsudik G, Tueche S, A Security Architecture for Computational Grids, *Proceedings 5th ACM Conference on Computer and Communications Security Conference*. p 83-92, 1998.
15. Jefferson Lab Replica Catalog, unpublished work, see http://www.jlab.org/datagrid/ [July 5, 2002] for additional information.
16. Java Web Start web site, http://java.sun.com/products/javawebstart/ [July 5, 2002].