

# An Integrated Global Service for File Transfer and Management in a Network (FTM)

Jim Almond and Peggy Lindner  
High Performance Computing Center Stuttgart (HLRS)  
University of Stuttgart  
Allmandring 30, 70550 Stuttgart, Germany  
saphir@citlink.net, lindner@hlrs.de

**Abstract**— *The management and access of distributed informational resources has emerged as an important requirement for scientific simulations using high-performance computing facilities. Both the input data and the output generated have reached levels exceeding the data management capabilities and the storage capacities of the researcher's workstation or other local facilities. We here describe FTM, a system deployed at the University of Stuttgart for the uniform, secure, interactive Access, Transfer, and Management of informational resources in a network or Grid consisting of facilities at distributed Virtual Organizations. The extension of this system to support computational workflows is also reported*

**Keywords**—file transfer, Grid environments.

## I. INTRODUCTION

The handling of scientific data in computational simulations has become an increasingly important issue. Emerging new technologies such as Grid computing [1], which give end-users the possibility to perform larger simulations than ever before, have increased the volume of the data sets which end-users have to deal with. The FTM system supports all necessary functionality for the access, transfer, and management of files and directories in a network or Grid environment, and can be considered as a replacement for - and enhancement of - ssh, scp, [4] and ftp [5] in a single unified system. Originally begun as a file-transfer component of the UNICORE [2] project, FTM has been enhanced and developed into an independent system. A fundamental goal of the system is the uniform and secure access of resources within a Virtual Organization or *Vorg*, using a Public Key Infrastructure (PKI) to remove the need for IP filters at firewalls, to provide uniform identification and authentication of users based on X.509 certificates, and to support access to multiple resources without the need to establish credentials by manually exporting certificates or using proxy servers. FTM also supports access to resources in the environment of the client itself.

The system is implemented in Java, using standard technology and off-the-shelf toolkits wherever possible. It is highly modular, allowing components to be interchanged with others of compatible functionality, and to be run on various local or distributed platforms.

FTM commands are executed in interactive mode, without the use of queuing mechanisms. Connection to remote resources is achieved using the Secure Socket Layer (SSL). In addition to the usual functionality offered by ssh, scp, ftp or gridFtp [3], FTM offers a number of enhancements, including selectable storage limits to protect servers and clients from mischievous or erroneous store requests, the specification of an "Overwrite Policy" to control the overwrite of existing files and directories, optional automatic backup of existing overwritten files or directories, optional control of compression and encryption during data transfer, optional verbose output, archival of files and directories as single ".zip" file, and third-party data transfer between remote storage servers. Both command-line (CLI) and GUI interfaces are available. Whereas the CLI supports only UNIX clients, the *NIFTI* (Network Interface to File Transfer in the Internet) GUI is available on both UNIX and Windows clients. Resource servers are currently only supported on UNIX.

Aimed at the requirements for uniform, secure access to the resources of a High Performance Computing (HPC) environment, FTM is not a typical "Gridware" project [1]. However, it could be incorporated into a Grid concept as a portal to HPC resources.

The organization of the paper is as follows: in section II we describe the functionality and specification of the FTM system. In section III we introduce the current available user interfaces, while section IV gives an overview on the system architecture.

As an extension to FTM, in section V we also introduce the *SAPHIR* project, which addresses the specification and execution of complex jobs, whose data and computational requirements are expressed in terms of workflow elements. Section VI finally gives a summary and an overview of the ongoing work.

## II. FUNCTIONALITY

### A. Global Specification of Resources

For the unique specification of resources in the global network, we define the following elements:

- A *Vorg* (Virtual Organization) represents a collection of

resources available to the user via a server at a given URI. Typically, a Vorg represents the facilities available at an organization or computation center at which the user is registered and has access rights defined by a user-ID.

- A *StorageServer* denotes a platform within the Vorg, on which requested resources are available. Each Vorg may contain any number of such StorageServers, some of which may also offer computational services
- A *Subspace* is a storage area (corresponding to a directory) within a StorageServer, specified either "seamlessly" by a SubspaceName such as "HOME" or by a SubspacePath such as "/users/home". The Subspace allows the emulation of concepts such as the UNIX Current Working Directory (CWD) and the idea of a "shortcut" to a commonly used storage area.
- The *pathname* is adopted from common usage on both UNIX and Windows platforms, and may be either a relative path within the subspace, or an absolute path. Wildcards in paths are supported to specify sets of matching resources

The values of Vorg, StorageServer, and Subspace taken together are termed a "Partner Specification" or *Pspec*, a kind of globalization of the UNIX concept of a working directory.

#### B. Global User Identity and Authentication

In typical production environments, the user is registered under a different user-ID at each Vorg, with access being protected by a (hopefully different) password for each user-ID. This is not only inadequate for secure user authentication, but also confusing and error-prone for the user. We solve these problems using Public Key Infrastructure (PKI) technology, providing the user with a reliable global identity in the form of an X.509 certificate. To use the FTM system, the user must possess a certificate, authenticated (signed) by a Certificate Authority (CA) recognized at the Vorg(s) to be accessed. This certificate accompanies the request and is used at the Vorg both to authenticate the user and to map the global identity to a local user-ID, using a database of known users and their certificates. The local user-ID is then typically used to control access to resources. Although the requirement for an X.509 certificate may place limits on immediate usage, we expect PKI technology to become established for the identification and authentication of potential HPC and Grid users.

#### C. Security Issues

As explained above, FTM establishes a secure SSL connection between the client and the server based on the certificates of the user and the server. To prevent server spoofing, the client also verifies the identity of the server. While this provides sufficient security for requests executed directly by the server, a problem arises when requests must be forwarded to another server, since this would require the "transitive trust" of all intermediate servers by the server at the final destination. To protect against possible corrupted or

"rogue" intermediates, the FTM system signs any requests which will be forwarded using the user's private key.

#### D. Data Management Functions

Management of remote files and directories is done by a set of functions corresponding to UNIX commands, but not bound to UNIX syntax. The functional spectrum currently includes listing, removing, changing access mode and group, creating directories, testing the properties of files and directories, renaming, etc. This functional palette is easily expandable. The user may control the level of information returned from the server by specifying verbose mode. For example, if the user requests the removal of *foo\** (wildcard expression), verbose mode will return a list of all files actually removed.

#### E. Data Transfer Functions

The system supports three essential transport services. *GET* copies files/directories from the (typically remote) storage of a *partner* to a specified location on the local client or on an alternative platform in the local environment of the client. *PUT* copies files/directories from the local client or from an alternative platform in the local environment of the client, to a specified (typically remote) partner location. *Transfer* copies files/directories from one remote storage location to another, either within the same StorageServer, within the same Vorg, or to another Vorg altogether (Third-party data transfer).

All data transfer functions support user-controllable options for the automatic compression of the transfer stream and its encryption for transfer across the public network. The overwrite of existing files is controlled by a user-selectable Overwrite policy, specifying whether existing files may be overwritten freely (as in UNIX), may never be overwritten, or whether an overwrite attempt aborts the process with an error exit. The user may also specify the automatic backup of overwritten files and directories. Transfer is accomplished using Java ZipStreams, avoiding any limitations to the size of transferred files.

#### F. Storage Navigation Functions

Analogous to the UNIX Current Working Directory (CWD) the FTM *Current Partner Specification* (CuPspec) represents the Vorg, StorageServer, and subspace to which the client is currently "connected", i.e. the (typically remote) partner storage space upon which commands will operate unless otherwise specified on the command line. Analogous commands *Fpwd* and *Fcd* allow the user to query or change the CuPspec.

### III. USER INTERFACES

Clients provide the interface to the user, relay requests to a specified (possibly remote) service, and return a result or feedback to the user. Two clients are currently available.

- 1) *NIFTI* is a GUI interface modeled after gFTP [6]. Figure 1 shows a screen shot of this interface. Given its modular

architecture, NIFTI is conceptually independent of the implementation of the services it represents, and currently interfaces alternatively to either FTM, scp or ftp services.

2) The Command Line Interface (CLI) is supported for UNIX clients. There are two variants:

a) In Single-command mode, an SSL connection to the server is established for each FTM command. This mode is useful for testing, and for the creation of UNIX scripts. Note that the prefix ":" is required for remote file paths in order to suppress the expansion of any wildcard characters by the client shell. Although a complete description of CLI is beyond the scope of this paper, the examples below should provide an idea of its capabilities.

```
[prompt]$ Fcd @:fs2/indat
```

Change CuPspec to connect to subspace indat within user's homespace at StorageServer fs2; the Vorg remains unchanged.

```
[prompt]$ Fls -v :foo?
```

Get a verbose (long) list of files foo? within CuPspec

```
-rw-r--r-- 1 myID myGroup 17440 Dec 15 22:43 foo9
-rw-r--r-- 1 myID myGroup 6939 Dec 15 22:41 foo3
-rw-r--r-- 1 myID myGroup 6939 May 22 22:41 foo1
```

```
[prompt]$ Frm :foo3
```

Remove foo3.

```
[prompt]$ Fget -ow 2 :foo9 :adir .
```

Copies foo9 and directory adir to the user's CWD. "-ow 2" specifies that any existing foo9 or adir are not to be overwritten.

b) Session Mode supports a single user sign-on, after which both UNIX and FTM commands can be issued over a single persistent connection to the server, in a manner analogous to ssh. This is more efficient, since SSL reconnect is avoided. No ":" prefix is necessary for remote file paths. The following commands illustrate the nature of the user interface:

```
[ :fs2/indat ]:>> ls -l foo?
```

Gets a long list of files corresponding to foo? within home/indat

```
[ :fs2/indat ]:>> pwd
```

Query the CuPspec on the partner side @134.56.14.44:fs2/indat (format is @VorgAddress:StorageServer/Subspace)

```
[ :fs2/indat ]:>> cd
```

Change to user's default Pspec on the partner platform (e.g. homespace on a favorite Vorg/StorageServer)

```
[ :fs2/HOME ]:>> lpwd
```

Query the CWD on the client side

```
[ :fs2/HOME ]:>> lcd localDir
```

cd to localDir on the client side

```
[ :fs2/HOME ]:>> get tenMB.file
```

Copy tenMB.file from home directory on the fs2 partner to localDir on the client

```
[ :fs2/HOME ]:>> Rm -v tenMB*
```

Remove files corresponding to tenMB\* from user's home on fs2, returning a verbose response reporting files actually removed

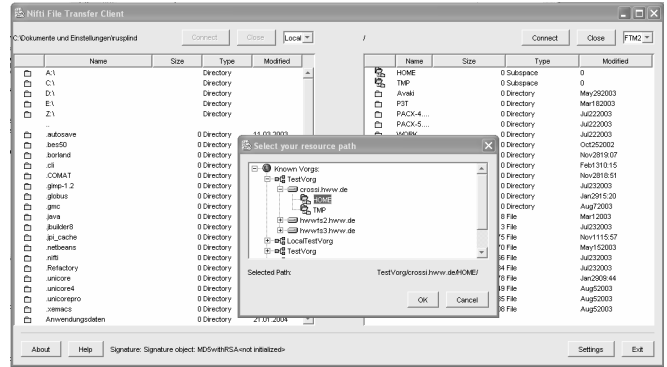


Figure 1. Screenshot of the NIFTI File Transfer GUI.

#### IV. SYSTEM ARCHITECTURE

As shown in Figure 2, the system architecture is comprised of three components and the protocols used to communicate among them.

##### A. System Components

The *CLIENT* initiates requests for actions or data transfer.

The *SAFE* (Secure Access File Executive) server runs at the Vorg, offering services on StorageServers to users whose certificates are registered in its local database. In the interest of security, SAFE should be installed on a dedicated platform running as a bastion host within the firewall network (if present), offering no services other than those required by SAFE, and containing no user accounts. Multiple servers can be employed. The actions of the server include:

- setting up the SSL connection to the client;
- receiving requests for services, and spawning threads for their execution;
- authenticating and identifying the user on the basis of the X.509 certificate sent with the request, and mapping it to a user-ID on the StorageServer;
- causing requests to be executed on the specified StorageServer in the name of the local user-ID;
- relaying byte streams for the transfer of files or directories; and
- returning a response to the client.

The *StorageServer* accepts commands (only) from its authorized SAFE server(s), and executes them in the name of the user. StorageServers must run the UNIX system, but do not require that Java be installed.

##### B. Inter-component communication

As shown in Figure 2, the connection between client and server is comprised of two segments:

1) The connection from the client to the SAFE server (typically across the public network) is currently an SSL socket built using the certificates of the user and the server. This single connection is used both for the exchange of request/response messages, and for the streaming of data for transfer operations. The use of SSL supports the (optional) encryption of the data stream as controlled by the user.

2) The connection from SAFE to a StorageServer is dependent on the individual requirements of the StorageAccessor. StorageAccessors have been implemented using rsh, ssh, and DCE access.

### C. Data streaming

All data transfer is done using streaming techniques, thus removing any limitations on the size of file which may be transferred, and obviating any temporary storage of files. The current implementation uses the Java ZipStream class for this purpose. However, the architecture supports the implementation of alternative stream types such as ftp streams, parallel streams, etc. The use of the ZipStream class supports selectable levels of data compression, and facilitates the transfer of multiple files and/or directories in a single request.

### D. Protocols and transfer mechanisms

FTM operations are performed in synchronous fashion. The client issues the request via a connection which remains intact until the operation is completed, and a final response is returned by the server. Commands such as *list* or *remove* use a simple request-response protocol. As depicted in Figure 3, the GET protocol operates in two-stages, in which the client first queries the availability of sources and assesses the viability of the request, and then requests and receives the available bytes. A final confirmation is necessary to prevent premature closing of the connection by the server. The PUT protocol uses a similar two-stage protocol.

As one might expect, there is a significant and useful dichotomy between Get and Put mechanisms. For a Get process, the server accesses storage to read files, producing a ZipStream, which the StreamStorer of the client stores using its StorageAccessor. For a Put, the roles are reversed; the client accesses storage to read files, and produces the ZipStream, which the server stores using its StorageAccessor. This dual use of many classes by clients and servers reduces the number of classes otherwise required.

### E. Performance

Data transfer performance is competitive with standard tools such as scp and ftp for incompressible files, and can be significantly superior to these when files are compressible. The option for unencrypted transfer can further enhance efficiency where appropriate.

## V. EXPANDING FTM TO SUPPORT COMPUTATIONAL WORKFLOW

The specification and execution of computational jobs can be considered to be a case of workflow technology, in which the actions in the workflow are essentially operations on and by files, often in a distributed environment. FTM thus offers an ideal foundation on which to build a distributed computational capability. The *SAPHIR* (Secure Access Portal to HPC Internet Resources) project exploits the capabilities of FTM to automate the specification and execution of complex

jobs. In *SAPHIR* we specify job complexes as a workflow in XML format employing a structure called a Work Request (Worq) based on the essential steps which typically comprise any computational operation.

- Create a workspace for the job on storage directly accessible by the computational platform.
- Assemble required input data into the workspace from various locations, including (possibly distributed) Storage Servers, and the user's client. In FTM terms, this involves requests to *PUT* data from the client, and requests to *transfer* data from (possibly distributed) StorageServers into the workspace.
- Execute one or more computational processes. These are essentially operations on executable files - possibly modified by application parameters - in which input files are used or consumed, and result files are produced in the workspace.
- Distribute specified result files to archive storage. In FTM terms, this involves requests to transfer results from the workspace to (possibly distributed) StorageServers.
- Return specified results to the initiator of the Worq (typically the original client). In FTM terms, this involves GET requests.

In order to support interdependent job steps, we assert that a dependency can arise only when a *successor* step requires, as input, one or more result files produced by a *predecessor*. This makes possible a simple data-driven dependency model which avoids the specification of a dependency graph such as in UNICORE [2]. This model is implemented by recursively supporting embedded Worqs within the data assembly phase of a Worq, with the returned results from the embedded Worq (the predecessor) providing the required input for the successor Worq. The current model allows a successor to be dependent on any number of predecessor Worqs, but limits a predecessor to a single successor, so that the job complex is limited to a strict tree structure. However, we believe this supports a majority of job complexes with reasonable effort. The actual control of the job flow simply uses the java thread mechanism.

*SAPHIR* is currently implemented as a client-centered workflow engine, which issues requests as specified in the XML job specification. All data assembly and result distribution operations are carried out interactively in session mode. Execution steps can either be done interactively or can be submitted to the batch subsystem of the computation platform. The client-centered model facilitates user interaction and the query of job status, but also requires the workflow engine to remain active for the duration of the entire workflow – even for batch steps. Future versions will likely remove this requirement by implementing a workflow engine on the server side.

## VI. CONCLUSION

This paper presented the concept and implementation of the

File Transfer and Management system FTM which offers an efficient, secure, and coherent system for the access of informational resources by distributed clients. Additional work is building on FTM to support the access of computational resources on behalf of HPC centers.

Currently the NIFTI interface is used with FTM as file transfer component in the Configuration Manager [7] in the DAMIEN [8] project at HLRS.

#### REFERENCES

[1] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, 1999.  
 [2] Dietmar Erwin (Ed.), *Joint Project Report for the BMBF Project UNICORE Plus*, Grant Number: 01 IR 001 A-D, Duration: January 2000 to December 2002, ISBN 3-00-011592-7, 2003.

[3] See <http://www-fp.globus.org/datagrid/deliverables/C2WPdraft3.pdf>.  
 [4] SSH protocol specification: <http://www.ietf.org/html.charters/secsh-charter.html>  
 [5] FTP protocol specification: <http://www.faqs.org/rfcs/rfc765.html>.  
 [6] gFTP project web page: <http://www.gftp.org>.  
 [7] Peggy Lindner, Natalia Currie-Linde, Michael M. Resch and Edgar Gabriel, 'Distributed Application Management in Heterogeneous Grids', in proceedings of the Euroweb 2002 conference, Oxford, UK, pp. 145-154, December 17-18, 2002.  
 [8] Edgar Gabriel, Rainer Keller, Peggy Lindner, Matthias Mueller, Michael Resch (2003) 'Software Development in the Grid: The DAMIEN tool-set', in P. M. A. Sloot, D. Abramson, A. V. Bogdano, J. J. Dongarra, A. Y. Zomaya and Y. E. Gorbachev(Eds.), *Computational Science - ICCS 2003*, Lecture Notes in Computer Science, vol. 2659, pp. 235-244, Springer.

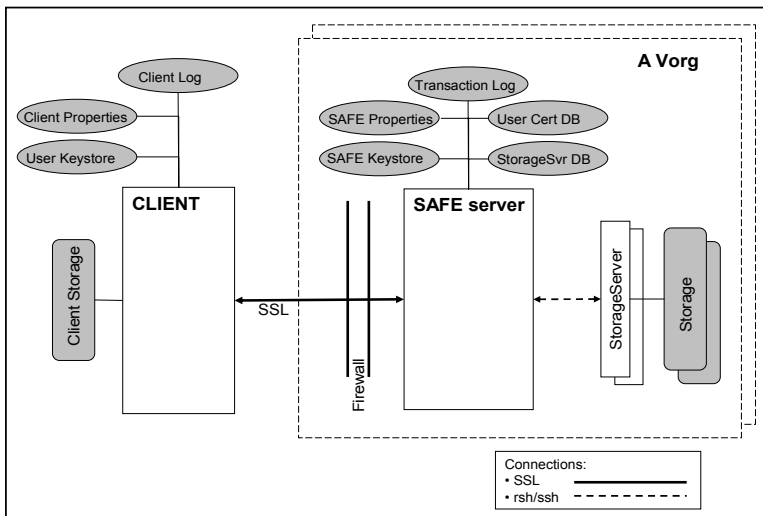


Figure 2. Basic Client-Server architecture.

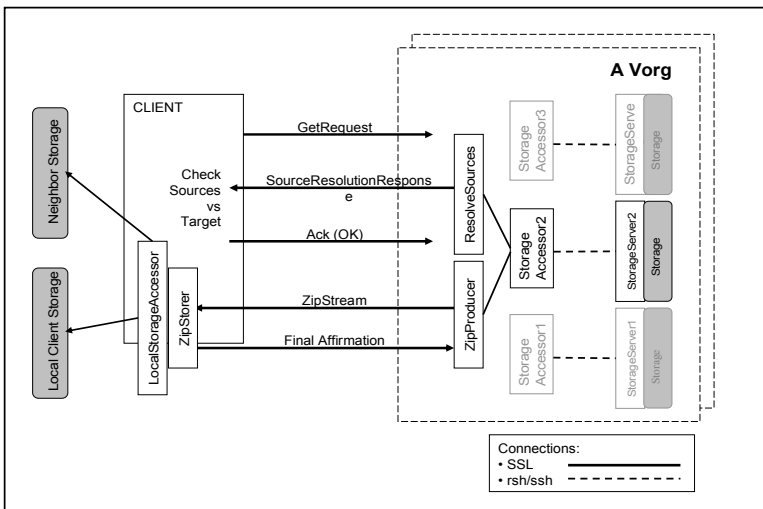


Figure 3. Simplified View of the GET protocol.