Technische Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr. Dr. Theo Härder

Diploma Thesis:

Comparative Evaluation of Implementations of Datagrids and Distributed File Systems

(SRB, EU DataGrid, AFS)

Peter Zimmer

November 14th, 2003

Tutors:
Dr. Ulrich Marder
Dr. Franz-Josef Pfreundt (Fraunhofer ITWM)

Erklärung des Autors

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur angefertigt habe.

Kaiserslautern, den 14. November 2003

Peter Zimmer

Abstract

In this diploma thesis, the concepts of *Distributed File Systems (DFSs)* and *datagrids* are shown and compared against each other. After comparison of the concepts, a suitable implementation is selected. The selected implementation is installed in an existing grid enviroment, the *Fraunhofer Resource Grid (FhRG)*.

First, the concepts of datagrids and DFSs are shown. After introducing the basic concepts, they are compared against each other. As the datagrid concept is the more global concept, the comparison results in the selection of the datagrid concept to be installed within the FhRG.

After selecting the datagrid concept to be installed within the FhRG, a suitable datagrid software has to be found. This is done by evaluating the two freeware datagrid implementations, *European Datagrid (EDG)* and *Storage Resource Broker (SRB)*. As no significant functional differences can be found, the SRB is selected with respect to non-functional aspects.

Before installation of the SRB within the FhRG, a concept of operation is developed to build a fast, reliable, scalable, and secure datagrid. This concept includes a global working concept to explain the placement and working parameters within the FhRG environment, an introduction to the *Grid Security Infrastructure (GSI)* authentication scheme, concepts for distributing and integrating the SRB, and a firewall concept that explains the most common firewall scenarios with their respective advantages and disadvantages. The installation procedure is described in detail.

To enable datagrid users to access the datagrid easily, a platform-independent client is developed. The client is a prototype to demonstrate the functions of the established datagrid.

Contents

Introduction

Distributed and data intensive computing gets the more and more important today. Traditional data management systems can not handle the huge amount of distributed data in an effective way. Thus, new approaches are made to handle the massive amount of data effectively. Two of these approaches to handle the distributed data are the *datagrid* and the *Distributed File System (DFS)* approach.

A *Distributed File System (DFS)* is an aggregation of distributed storage resources. Several distributed nodes within the DFS system share an arbitrary number of local directories with the other DFS nodes. The DFS system abstracts the physical storage resources to span up a uniform namespace. Thus, DFS clients are able to access the DFS resources in a uniform way. The inspected DFS implementation is the *Andrew File System (AFS)*.

A datagrid can be seen as a generalized DFS that guarantees worldwide secure data access to its users. Usually, a datagrid consists both of several distributed storage systems, the datagrid nodes, and of management servers, the metadata servers.

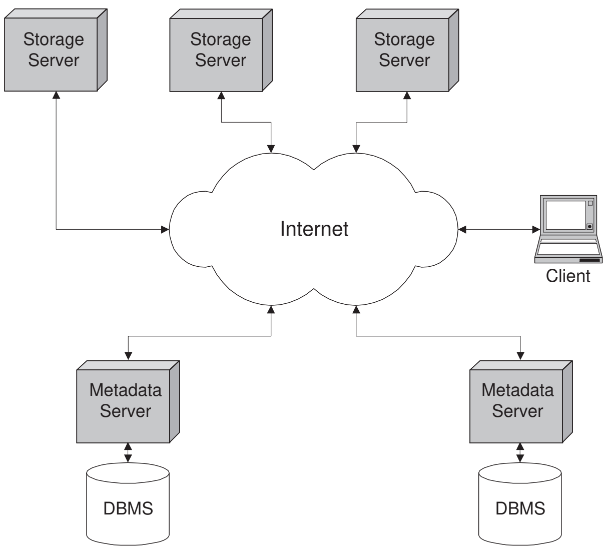

Figure 1: Datagrid Scheme

The storage systems are responsible for storing and transferring data, while the metadata servers are responsible for the complete metadata management. This includes management of user authentication and authorization, management of file replica and file synchronization, and storage resource management.

Data access within a datagrid is done in a uniform way, thus datagrid users do not need to know where the data is located. A datagrid software manages replica of user data to speed up the file access time.

Known datagrid software implementations include the *European DataGrid [1]*, *AVAKI DataGrid [2]*, the *Storage Resource Broker [3]*, the *Particle Physics Data Grid [4]*, the *Grid Physics Network [5]*, and the *China Clipper [6]* project.

The main instance to control and administer the grid development activities of the different grid development groups is the *Global Grid Forum (GGF)*. The *Global Grid Form (GGF)* [7] is a group of individuals engaged in research, development, deployment, and support activities related to high-capability distributed software system, or *Grids*. It was formed in June 1999 under control of the *Grid Forum*. As the GGF has a wide spectrum of tasks, the tasks are subdivided into *areas*. Within the GGF, *working groups* and *research groups* were formed to handle the various tasks.

An *area* aggregates some of the working groups and research groups. The GGF is managed by a general chair, a steering group, and working group and research group chairs. The GGF chair is responsible for ensuring that the GGF produces coherent, coordinated, architecturally consistent, and timely output as a contribution to the overall evolution of *grid* architecture and technologies.

The GGF defined the *Open Grid Services Architecture Platform (OGSA) [8]* as a *vision of a broadly applicable and adopted Framework for distributed system integration*. OGSA defines an enabling infrastructure for systems and applications that require the integration and management of services within distributed, heterogenous, dynamic organizations. The OGSA core is the *Open Grid Services Infrastructure (OGSI) [9]*.

OGSI defines important parts of a distributed system, including standard interfaces and associated behaviors. It defines mechanisms for managing, creating, and exchanging information between Grid services. To access an OGSI compatible system, the system offers a *Web Services Definition Language (WSDL) [10]* interface.

The main tasks of this diploma thesis are the comparison of the datagrid concept with the classical *Distributed Filesystem (DFS)* concept, the comparative evaluation of the datagrid implementations EDG and SRB, the selection of one of the evaluated datagrid implementations, and finally the integration of the selected implementation in an existing computing grid infrastructure, the *Fraunhofer Resource Grid (FhRG)*.

To accomplish these tasks, some prerequisites will have to be made. The first chapter describes some datagrid basics, introduces the *Fraunhofer Resource Grid (FhRG)* as the target platform for the datagrid software, and finally works out the analysis criteria for the comparative evaluation. In chapter 2, the concept of datagrids is compared to the concept of AFS, a classical Distributed File System. Chapter 3 deals with the comparative evaluation of the two datagrid implementations *European DataGrid (EDG)* and *Storage Resource Broker (SRB)*. In chapter 4, a concept of operation is developed. This concept covers the steps to distribute and integrate the datagrid software within the existing *FhRG* infrastructure. Chapter 5 describes the necessary steps to install the selected datagrid software in the *FhRG*. The chapter 6 deals with the development of a stand-alone datagrid client. In Chapter 7, a short summary of this thesis and an outlook of what to come in the wide area of datagrids is given.

Chapter 1

Datagrid Conceptual Overview

This first chapter describes the basic concepts of datagrid operations. First, an introduction to the operating concepts of a datagrid is given. The introduction is followed by a description of the *Fraunhofer Resource Grid* as a potential datagrid environment. The chapter is finished by a definition of evaluation criteria for a datagrid software. These criteria will be used in chapter 3 to evaluate the datagrid software.

1.1 Introducing Datagrids

This section describes the basic concepts of a datagrid architecture. Starting with the explanation of the structure of a datagrid architecture. The structure description is followed by a short procedural description.

A datagrid architecture consists of some distributed data servers that store the data within the datagrid and at least one metadata server that manages the necessary metadata of the datagrid.

The data servers are responsible for management and distribution of data within the datagrid. They guarantee worldwide secure and confidential data access to all datagrid users. All data servers exchange data among each others through different file-transfer mechanisms. The datagrid metadata is managed by at least one metadata server.

A metadata server is responsible for user authentication and authorization, file synchronization operations, and the metadata management in general. It stores metadata about the datagrid system which includes account information, file status information and resource information.

If a datagrid user wants to access the data stored in the datagrid, it connects to a login node (login node can be a data server, metadata server, or any other special node) and sends its login data. The login node validates the submitted login information at the metadata server. If the user gets successfully validated, the login node authenticates this user for the usage of this datagrid system.

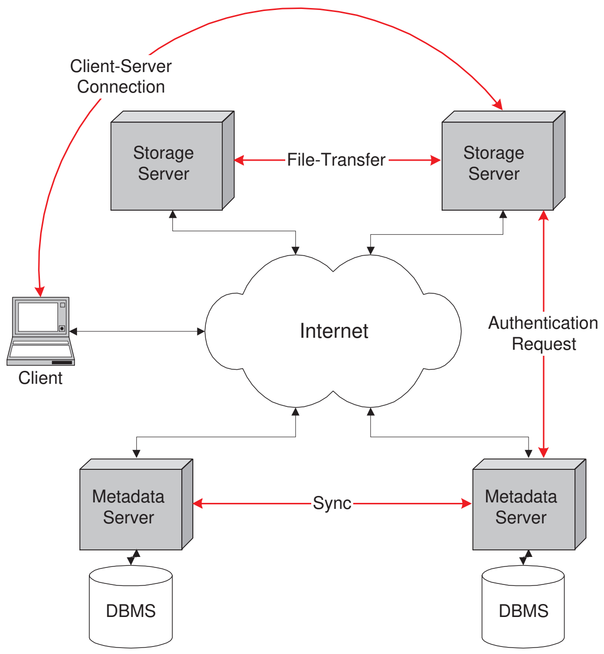

Figure 1.1: Datagrid File-Transfer Scheme

One major task of a datagrid architecture is the replica management. To speed up file access, users can instruct the datagrid node to copy data to another datagrid node which offers faster data access. The performance of data transfers is a value that depends on the users location.

The data representation is done in a uniform way within the datagrid, thus the user can access all data in a similar fashion. If a user wants to embed the datagrid access mechanism in his own client application, usually a datagrid application offers an *Application Programmers Interface (API)* that provides of all the necessary functions to access a datagrid.

Figure 1.1 depicts the flow of information within a datagrid. The red arrows show the logical connections, the black arrows describe the physical connections. The client connects to a data server and requests authentication. This data server asks the metadata server for client authentication. If the metadata server grants user authentication, the data server authenticates the user for usage of the datagrid system. If the user requests a file operation on files that are not located on the login node, a file-transfer with another data server occurs.

1.2 Fraunhofer Resource Grid

The *Fraunhofer Resource Grid (FhRG)* is an aggregation of hardware-resources and software-resources of five participating Fraunhofer institutes. The goals of the FhRG are the provision of bundled compute power and many applications to industry, development acceleration, generation of massive compute power, and it gives software-developers a platform to offer their specialized software components.

The FhRG infrastructure is based on underlying globus grid services software (the Globus Toolkit) which offers a complete set of functions to get low-level access to a Grid. FhRG builds a middleware that offers a high-level interface to the Grid. X.509 [11] certificates are used to identify grid-users. To provide secure and confidential communication, the complete communication with the high-level interface is done by the *Transport Layer Security (TLS) [12]* mechanism. Figure 1.2 depicts the security architecture of FhRG:

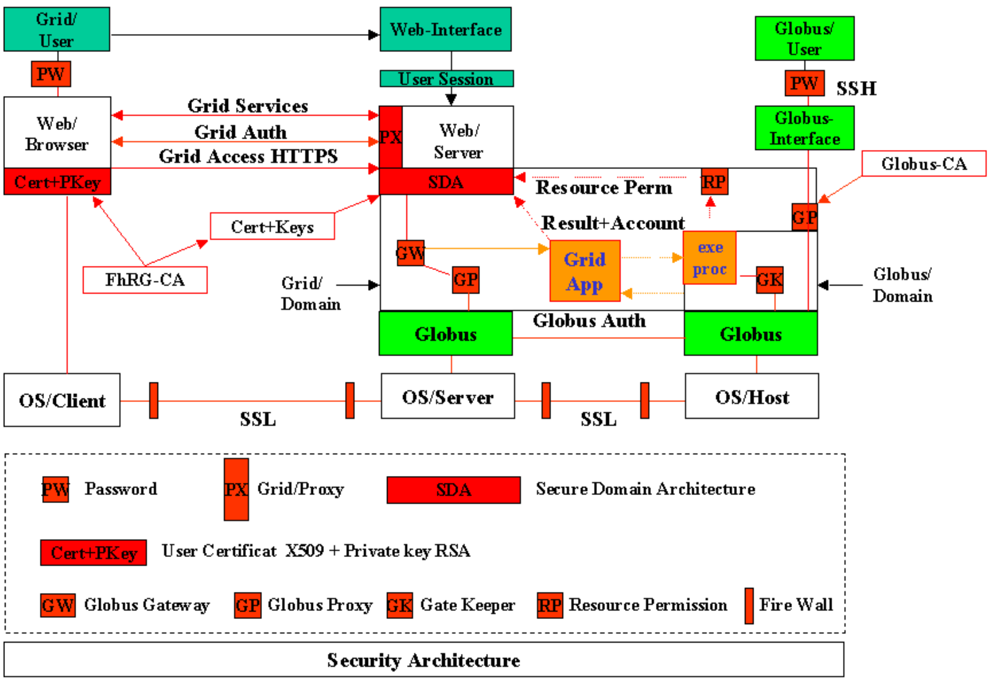

Figure 1.2: FhRG security architecture (Source: [13])

To exchange both program data and hardware metadata with the corresponding grid services, an XML-based [14] description language was developed, the *Grid Application Definition Language (GADL)* [13].

The FhRG in its current state lacks of a datagrid. This software has to offer some functionality:

- An API to easily integrate the datagrid software within the FhRG.
- Interaction with existing storage resources.
- Secure and confidential communication.
- Support for X.509 certificates.

- Uniform namespace to manage the files within the datagrid in an efficient way.
- multi-user capability
- Fast file transfers to minimize overhead.

One task of this diploma thesis is to install and integrate a datagrid software to the existing FhRG computing grid. In the next chapter, the concept of classical DFS is compared to the concept of a datagrid. Chapter 3 gives an evaluation of the two datagrid implementations EDG and SRB with a selection of the software that meets the defined requirements best.

Before the software can be installed and integrated in the FhRG, a concept of the future software use has to be developed. This is done in chapter 4. Chapter 5 describes the installation and integration procedure itself.

1.3 Datagrid Analysis Criteria

The given working environment shows up some requirements that have to be fulfilled by a datagrid software. To evaluate datagrid software respecting to this requirements, analysis criteria for the datagrid software are defined.
From the requirements of FhRG the following analysis criteria are reasonable:

1. **Architecture:**
   The criterion *Architecture* includes a short conceptual overview of the datagrid architecture, a datagrid component description, and a description of the component interaction in a running datagrid environment.

   The architectural overview introduces the reader to the basic structures of both datagrid implementations.

2. **Namespace:**
   A datagrid generates its own namespace to represent the stored files to the client software. A datagrid stores files on heterogeneous media that can not be accessed in a uniform way. To give the datagrid users a uniform view to their data, a datagrid abstracts uniform namespace from the heterogeneous storage media namespaces.

   The namespace concept is very essential within the concept of a datagrid, thus, the analysis of the generated namespace seems to be very reasonable.

3. **Storage Media Abstraction:**
   To use its attached storage in a more efficient way, the software abstract their managed files from the attached storage hardware.

   The files stored in a datagrid are of very different size. To optimize the file-transfer speed, a datagrid might optimize the file-transfers by packaging many small files into one large file that is stored much faster on a slow storage media like a tape.

Another example is to implement a caching hierarchy that speeds up the file-access or to automate replica generation of files through the appropriate storage volume structures.

4. **Caching:**
   To speed up datagrid I/O transfers, some of the software implement a caching structure within every single datagrid node. The idea of caching in datagrids is to pipe every I/O transfer in a datagrid node through a fast caching system.

   Within the caching hierarchy, the caching storage media are fast, but small, storage media that are accessed directly from the client. The permanent storage resources are large, but slow, storage media in the backend that can not be accessed directly from the client application. With this hierarchy, file-transfers can be speed up.

5. **Replica Management:**
   To work in an efficient way, a datagrid uses dataset replicas. If a user wants to use its dataset at location A and the original dataset is located at B, the datagrid generates a replica of the dataset at location A to give the user faster access to his dataset. If the user alters the replica of the dataset, the replica and the original of the dataset are out-of-sync and need to be synchronized.

   The management of file replicas is an essential point in the datagrid construct, thus, the replica management of the two datagrid implementations has to be compared each other.

6. **User Management:**
   A datagrid is a multi-user environment by definition. It consists of multiple nodes each with a separate user authentication, which is necessary for security reasons.

   The user-management feature should support grouping of users into independent organizational units. Thus, many users can be managed very easy by the datagrid administrators.

   As shown above, the FhRG requires a single sign-on for every datagrid user. To fulfill this requirement, each software has its own concept.

   For security reasons, the complete authentication procedure should be encrypted. Thus, no intruder are able sniff user passwords.

7. **File-Transfer:**
   To bring data (files, datasets, etc.) from one point within the datagrid to another point in the datagrid is the main task of a datagrid.

As all datagrid tasks depend on the file-transfer mechanisms of the datagrid, a fast and reliable transfer is necessary. Some datagrid software use the *GridFTP [15]* software from the Globus Group, while others use proprietary solutions.

In addition, security aspects might be important. If the datagrid requires a high security level, an encrypted file-transfer communication has to be supported.

8. **API:**
   To allow datagrid users to access the datagrid, each datagrid software offers an API. An API is necessary to access a datagrid and is useful to integrate the datagrid usage in own software.

   It has to support all the necessary functions to manage files stored in the datagrid to give the datagrid users the ability to access data stored in the datagrid. The management functions have to include functions to read, write, and replicate files.

Chapter 2

Datagrids and Distributed File Systems

This chapter compares the concepts of the datagrid architecture with the concepts of the classical *Distributed File System (DFS)* architecture. The first section introduces the concepts of the *Andrew File System (AFS)*, the most used *DFS* worldwide. In the second section, the concepts of *datagrids* and *DFSs* are compared against each other.

2.1 Andrew File System

The *Andrew File System (AFS)* is a distributed filesystem product, pioneered at Carnegie Mellon University and supported and developed as a product by Transarc Corporation (now IBM Pittsburgh Labs).

It offers a client-server architecture for file-sharing, providing location independence, scalability and transparent migration capabilities for data.

There is also an open source implementation of AFS available called OpenAFS [16]. It is developed from the AFS source tree released by IBM.

Namespace

AFS uses a uniform namespace with strong abstraction of the storage hardware from the filename. It uses a client/server computing model where each machine is able to act as a client or as a server.

The AFS server stores data and performs services for clients. AFS clients access data stored on AFS server machines and perform computations for users.

AFS acts as a classical *Distributed File System (DFS)* which joins together the file systems of multiple server machines. This architecture makes it easy to access data stored on remote server machines.

AFS hides its distributed nature. Data shared by the AFS servers can be mounted by any AFS client in the local directory structure, thus, accessing the distributed files is exactly the same as accessing local files.

Administration

The administrative domain within the AFS is the *cell*. Each cell has at least one *cell administrator* which is responsible for data storage on the multiple storage servers and how client machines of this cell are configured. A cell can be seen as a grouping of client and server machines defined to belong to the same cell. The cell to which your local machine belongs to is called the *local cell*. All other cells are called *foreign cells*.

AFS sites are groupings of one or more cells. For example, all cells of the ITWM Corporation form a single site.

Security

AFS uses a password-based authentication method with mutual authentication.

To authorize the users with the AFS file system, *Access Control Lists (ACLs)* are used. Each AFS directory has an ACL that contains information what actions from what users are allowed on the files and directories contained in this directory.

Owners of a directory and cell administrators can always administer an ACL. New users own their home directory including sub-directories automatically. Non-owner users are only able to administer an ACL if and only if they got the appropriate permission from an owner user for this directory.

Multiple users can be aggregated in a group. A group can have an entry in the ACL of an arbitrary directory, the group permissions are granted to all users that are members of this group.

File Access

An AFS client can access the data stored in the AFS file space only through the *cache manager*. The cache manager is a part of the AFS clients kernel that communicates with the AFS server processes running on the AFS storage servers. It translates file requests made by a client application into *Remote Procedure Calls (RPCs)* to the appropriate file server that contains the required data.

To determine the appropriate file server, the cache manager contacts the *Volume Location (VL)* service that tells the clients cache manager where the requested files are located. If the call was successful, the file can be downloaded from the file server.

Each requested file that was successfully downloaded from a file server is cached locally in the clients AFS file space. This aggressive caching speeds up the working on remote files. On the other hand, problems with file inconsistencies may occur.

To avoid file consistency problems, the AFS file server sends a *callback* along with the file itself to the clients cache manager every time the clients cache manager requests a modifiable file. If the master copy of a modifiable file is changed, the callback is broken by the file server. When the clients cache manager requests data from this file, it recognizes the broken callback and requests the new version of the file.

To save changes made in a modifiable file, the client application must call a save command. Like in the UNIX file systems, the most recent version of a file is taken. If two clients are modifying a file, the last client that saves the file, updates the master copy on the file server permanently.

If an AFS file server becomes unaccessible, the client can continue its work on the local cached files. As the file server is inaccessible, it accepts no save commands. Once the file server is online again, every client is able to save the work as usual.

Storage Organization

The storage within an AFS file server is divided into sections called *partitions*. Partitions are subdivided into units called *volumes* that contain a subtree of directories and files. Figure 2.1 depicts the organization of storage hardware within the AFS system.

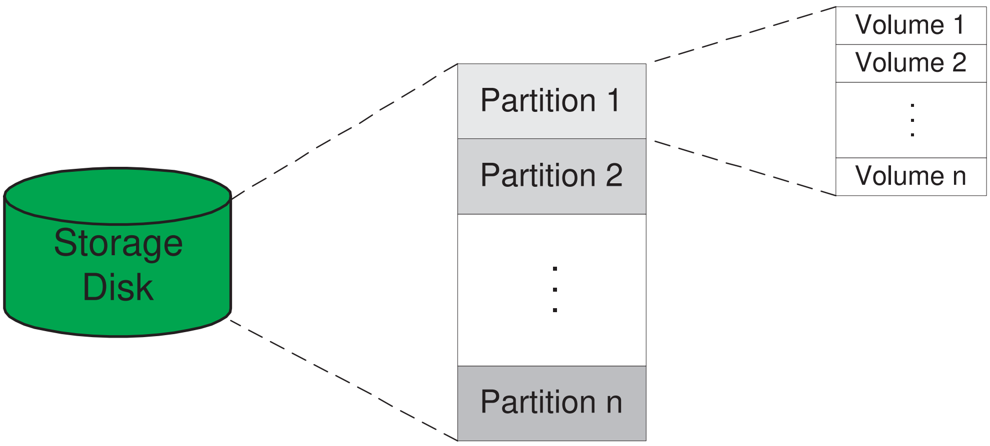

Figure 2.1: AFS Storage Organization

As AFS automatically tracks the locations of its volumes, system administrators can move volumes from one file server to another file server without interrupting clients work.

AFS volumes are mounted within the AFS file space on the client machine. Thus, client applications can access data through the AFS mount points.

If a file server becomes inaccessible, only the volumes stored on this file server become unavailable. All other volumes are not affected by the breakdown of one single file server.

AFS supports the use of *quotas*. System administrators can set a size limit (quota) that describes the maximum amount of disk space a volume can consume.

If a client application tries to exceed this quota, it receives an error message. Volumes have completely independent quotas. The quota of one volume does not affect the quotas of the other volumes. The total of all volume quotas within a partition should be less or equal the partitions size.

2.2 Comparison of Concepts

The two concepts of datagrids and Distributed Filesystems are very similar in many aspects. This section points out some of the essential equalities and shows up the most important inequalities between the two compared concepts. At the end of the section, a view to the main intended working area (working focus) of each concept is given.

All considerations made in this section assume a datagrid implementation with all standard features enabled and the DFS implementation AFS described above.

Comparison
The main task of both architectures is to give the users the ability to access their distributed data easily. Both concepts support the data access on multiple distributed storage servers.

One of the main aspects of both concepts is the realization of a uniform namespace to make the usage by the clients as easy as possible. An authorized client sees a directory-like file structure without any knowledge of the physical storage location of any files. The physical resources are *abstracted* from the clients view.

Both concepts support user management mechanisms that enable the site administrators to manage their users in an efficient way. Regarding the user authentication, both concepts support strong encrypted user authentication, ACL based user authorization and user grouping with policy management.

The connections necessary for user authentication and data transfer between storage servers and the data transfer between storage servers and client applications can be encrypted for security reasons in both concepts.

Up to this point, both concepts seem to have similar goals, but there are differences between the concepts. These inequalities are shown in the next paragraphs.

To access resources shared by AFS from a client that runs on a UNIX-like operating system, the clients kernel must be patched. As AFS operates on mount points, this kernel patch is necessary.

If the client runs on the windows operating system, some AFS-specific libraries have to be installed. As the AFS uses standard mount points, accessing the system from a client is quite easy. The client accesses the AFS mount point exactly like any other local mount point. If the storage server associated with a mount point becomes unaccessible, only the mount point becomes inaccessible, the other AFS mount point are not affected.

A typical datagrid realization requires no special kernel or system modification. The system is accessed through an *Application Programmers Interface (API)*. As this is no standard API, special clients have to be written to be able to access resources shared by a datagrid system. A datagrid client development procedure is described in chapter 6.

As AFS acts as a very transparent application that offers standard UNIX mount points, there is no need to assign metadata to the data stored in the AFS system. The only metadata that is necessary for the AFS system is user-management related metadata like the ACLs.

In datagrids, metadata is an essential component. Many different types of metadata are assigned to datasets in the datagrid system (for example metadata regarding the file replica, user definable metadata, etc.). As much as possible information is cut off from a file and transferred in the metadata of the file. Mostly, the metadata is stored in a separate database that is located on a non-storage node of the datagrid. The computer that runs the database should only serve database requests, no storage operations should occur on this *metadata server*.

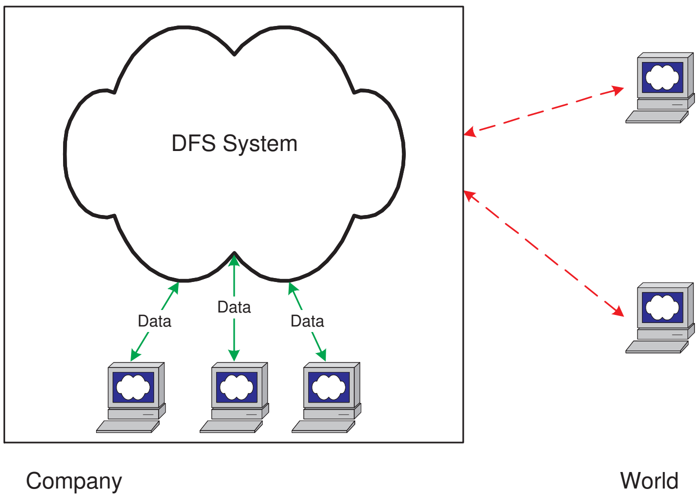

Figure 2.2: AFS Conceptual View

Summary

Figure 2.2 depicts a conceptual view of a typical AFS realization. Computers within the company that runs the AFS system are allowed to access resources shared by AFS. All other computers outside the company can not access the AFS resources. The distributed file system concept of AFS can be called a *closed system* which allows internal access from within the administrative cell but interdicts external access from the outside world.

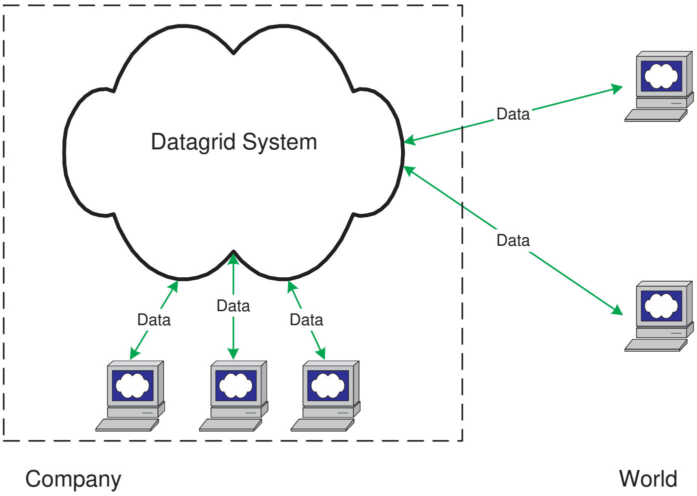

Figure 2.3: Datagrid Conceptual View

Figure 2.3 shows a conceptual view of a typical datagrid realization. All computers within the company that establishes a datagrid are able to access the datagrid resources. Computers outside the company are also allowed to access the datagrid resources according to their respective permission. A datagrid is an *open system* that allows both access from inside the administrative cell and from the outside world.

Due to its nature as a *closed system*, the AFS does not support any special storage devices in the backend. A typical datagrid realization supports special storage devices in the backend, for example tape-pools, databases with large object support, or HPSS.

	AFS	**Datagrid**
Access Method	Linux Mount Point	API functions
Uniform Namespace	Yes	Yes
User Management	Yes	Yes
Metadata	No file-specific	Yes
Supports non-standard storage	No	Yes
Strong security	Yes	Yes
OS patch necessary	Yes	No
Intention	Closed System	Open System

Chapter 3

Datagrid Evaluation

In this chapter, two datagrid implementations are being presented, analyzed and finally compared against each other.

First to come are the presentations of the *European DataGrid* and *Storage Resource Broker* software. Subsequent to the presentations, there is an analysis of the software with respect to the previously defined analysis criteria.

For each analysis criterion, the implementations are presented under careful attention of the respective criterion. At the end of each criterion analysis, the two software are compared against each other.

At the end of the chapter, one software is being selected with respect to the analysis results. The selected software will be installed within the *Fraunhofer Resource Grid (FhRG)* environment.

Introducing European DataGrid
The first software to be analyzed is the open-source software *European DataGrid (EDG)*. The EDG is a project funded by the European Union with the aim of setting up a computational and data-intensive grid of resources for the analysis of data coming from scientific exploration. The project spans from 2001 to 2003 with over 200 scientists involved under the leadership of CERN, the European Organization for Nuclear Research. The EDG software is to be seen as under extreme development. It will be continued in the *Enabling Grids for E-science in Europe (EGEE)* project funded by the European Union.

All descriptions about European DataGrid are taken from [17, 18, 19, 20].

Introducing Storage Resource Broker
The second analyzed software is the *Storage Resource Broker (SRB)* implemented by *San Diego Supercomputer Center (SDSC)*. SDSC is the supercomputing department of the University of San Diego, CA. SRB is a client-server middleware that provides a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated data sets. SRB v1.0 was released in 1997 as one of the components that result from SDSC's *Massive Data Analysis System*

(MDAS) project in collaboration with IBM. In 1999, SRB was installed at 14 sites around USA. The sites include SDSC, Caltech, UC Berkeley and NCSA. The current version 3 of SRB was released October 1st, 2003.

The SRB tool is working in many different projects, including astronomy, ecology and environmental sciences, medical sciences, molecular sciences, physics, chemistry and many more. In total, SRB is established and working on more than 50 sites. All descriptions about SRB are taken from [21, 22, 23, 24, 25, 26].

3.1 Architecture

This section gives a conceptual view to the architectures of SRB and EDG. The main concepts of both datagrid realizations are presented to give the reader an overview of the software that is being compared in the subsequent sections.

3.1.1 European DataGrid

In this section, the architecture of the *European DataGrid (EDG)* datagrid software will be presented. First of all, the components of EDG will be shown and in the following the components of EDG will be explained.

3.1.1.1 EDG components

The *EDG* consists of four main components:

- **Storage Resource Manager :**
  The *Storage Resource Manager (SRM)* is a high-level interface to a storage system which manages several storage devices (hard disk, SAN, tape silo).

- **EDG Replica Manager :**
  The *EDG Replica Manager (ERM)* is an interface to all the metadata-management operations. This interface is used by EDG clients to manage the metadata of their file-objects.

- **Grid Data Mirroring Package :** The *Grid Data Mirroring Package (GDMP)* is a package responsible for all file-transfer operations. It can be used in conjunction with the ERM described above.

- **Client :**
  The EDG Client connects to EDG to use the services of the EDG system.

The SRM Server manages its attached storage resources. To get efficient I/O operations on the storage resources, at least one SRM server has to be installed on all EDG sites.

The ERM manages all metadata operations. It consists of the components *Replica Location Service (RLS), Replica Metadata Catalog (RMC), Replica Storage Handler (RSH)* and *Replica Optimization Service (ROS)*.

Within the EDG, each file-object is identified by a unique *Storage File Name (SFN)*. The SFN is a locator for a physical file, where the scheme specific part is understood by the *Storage Resource Manager* service. It is a *Uniform Resource Locator (URL)* where the scheme is *SFN* and the host is a valid *Storage Resource Manager* host. At registration time of the SFN, a *Grid Unique Identifier (GUID)* for that SFN is generated. The GUID is a immutable and grid-unique identifier for a file-object. It is a 128-bit number which is guaranteed unique until 3400 A.D.

A GUID is a cryptic identifier that is not intended to be human-readable. A user can create some alias names for each GUID. These alias names are called *Logical File Names (LFN)*. An LFN should be globally unique, but the *Replica Management System (RMS)* service can not control the uniqueness.

To translate an LFN or GUID into an SFN, the Metadata and Replica Location Catalogs must be contacted. The returned SFN is a *Universal Resource Identifier (URI)* having the corresponding SRM as its hostname. The contacted SRM returns a *Transport URL (TURL)* to the client application that contacted the SRM. This TURL provides all the required information to access the file-object (for example protocol, host, port and path).

An EDG-based datagrid consists of one or more *Storage Elements (SEs)* that store all the data managed by this datagrid. Each SE is represented by a node in the EDG datagrid.

GDMP is an EDG package to handle all the file-transfer operations in an efficient way. The *GDMP* can be used to transfer files from one SE to another SE within the EDG system. GDMP is described in detail at section 3.7.

The subsequent paragraph *EDG Replica Manager* describes the ERM structures in detail.

EDG Replica Manager The *EDG Replica Manager (ERM)* is an interface to all *Replica Management Services (RMS)* operations. These RMS operations can be summarized as:

- **Replica Metadata Catalog :**
  The *Replica Metadata Catalog (RMC)* stores LFN aliases for given GUIDs and the corresponding LFN and GUID attributes.

- **Replica Location Service :**
  This distributed service provides the mappings between GUIDs and SFNs. It is contacted to get a fully-qualified SFN for a given GUID. The RLS consists of two components, the *Local Replica Catalog (RLC)* and the *Replica Location Index (RLI)* :

- **Local Replica Catalog :**
  The *Local Replica Catalog (LRC)* stores the GUID to SFN mappings for a single site or a single *SRM* at a site. It stores only the mappings for one given SRM.
- **Replica Location Index :**
  The purpose of the *Replica Location Index (RLI)* is to link the LRCs together. This catalog stores the GUIDs with the associated LRC. If a mapping-request arrives, the RLI provides the LRC which stores the mapping for the given GUID.

- **Replica Storage Handler :**
  The *Replica Storage Handler (RSH)* manages the automatic replication between *SRM*s for specified datasets.

- **Replica Optimization Service :**
  This service intends to evaluate and minimize the cost of I/O transaction. It is still under heavy development and so it will be neglected in this diploma thesis.

3.1.2 SRB

This section explains the architecture of the SDSC *Storage Resource Broker (SRB)*. First of all, some SRB key-features are presented. Subsequent to the key-features, the components of the SRB are shown.

3.1.2.1 SRB Features

The major features of SRB include:

- SRB Server:
  The SRB Server is a middleware that provides clients a uniform access to heterogeneous storage media over a network. The SRB Server will be described in detail in paragraph 3.1.2.2.

- MCAT Server:
  All the metadata management for SRB is done by the *Metadata Catalog (MCAT)* Server. The MCAT Server uses a conventional relational database management system like Oracle, DB/2, Sybase or PostgreSQL. The MCAT server will be explained in paragraph 3.1.2.2 later in this section.

- Zones:
  SRB supports the usage of *zones*. Zones are organizational units that represent a single organization. Each zone has exactly one MCAT server that manages all the metadata-related tasks within this zone. Between related

MCAT servers, inter-zone transfers are possible. This is a completely new feature in SRB 3.0, thus, the zones feature is still under heavy development.

- Global Logical Namespace:
  Within SRB datagrid, the file-objects are organized in a directory-like structure. Several file-objects can be aggregated into one *collection*. The collections are arranged in a hierarchical manner, thus, a tree-like structure is developed. Details of the namespace problem area are described in section 3.2.

- Container:
  Due to the high overhead of file access in archival storage systems, the performance of accessing small files from an archival storage system is very poor. SRB is able to aggregate small files in the cache to a *container*. The *container* abstraction speeds up the performance of accessing small files in an archival storage system.

  All caching concepts including the *container* mechanism are discussed in detail in section 3.4.

- Data Replication:
  SRB supports both the automatic and the manual replication of data. The data replication is discussed in detail in section 3.5.

- Replica Synchronization:
  The data replica can be modified individually so that the replicas of a file-object can be out-of-sync. SRB supports the synchronization of replica. In section 3.5 the problem area of replica generation and replica synchronization is discussed in detail.

- User Authentication:
  SRB supports six authentication schemes:

  - plain password
  - encrypted password
  - SDSC Encryption / Authentication (SEA)
  - SEA encrypted
  - Grid Security Infrastructure (GSI)
  - GSI encrypted

  All these authentication schemes and other aspects of the user authentication/authorization problem area are discussed in section 3.6.

- Tickets:
  SRB supports the *ticketing* mechanism. Datagrid users are able to grant read access to their data both to other datagrid users and foreign people by issuing a ticket to the corresponding person.

  This is discussed in paragraph *Tickets* on page 35.

- Mass Storage System :
  Through the use of a *Mass Storage System (MSS)* datagrid users are able to define a set of caching resources in the frontend and a set of archiving resources in the backend. This caching system is described in detail in section 3.4.

- API:
  Clients can contact the SRB Server via an API. The API description is in section 3.8.

- Large File Support:
  Since version 2.0.0, the SRB Server supports files larger than 2GB.

- Monitoring Daemon:
  srbMon is included in the SRB package. It is a monitoring daemon that remotely monitors and logs a set of distributed SRB servers. In case of failure it tries to restart the broken SRB Server.

- Auditing:
  Auditing provides a facility to audit the usage of data-objects.

- SDSC Matrix:
  The *SDSC Matrix* is a service (SOAP [27] /WSDL [10]) oriented architecture of *SRB*. It delivers multi-organizational information storage management services.

- Supported Platforms:
  SRB supports the Operating Systems AIX, Sun Solaris (32 and 64 Bit), Linux (32 and 64 Bit), SunOs, Dec OSF, SGI, MacOS X, Cray C90 and Microsoft Windows 2000/XP.

  The MCAT Server is able to communicate with the Oracle, DB2, PostgreSQL and Sybase DBMS.

3.1.2.2 SRB Components

An SRB-based datagrid consists of one or more zones while each zone aggregates exactly one MCAT-server, one or more SRB storage servers, and some SRB clients. Figure 3.1 shows a simplified view of the SRB middleware where one SRB client, one SRB server, and the MCAT server in the zone *ITWM* are presented.

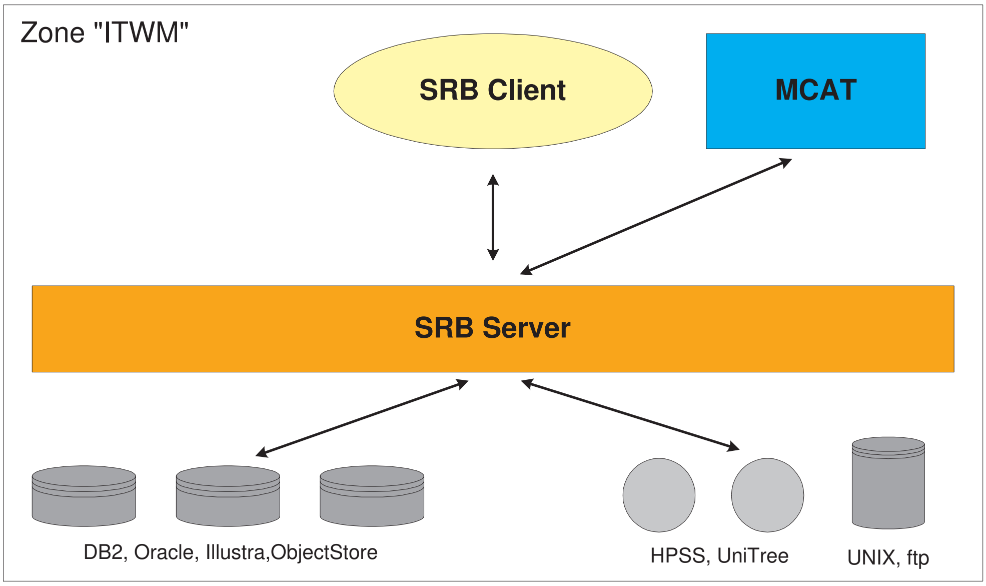

Figure 3.1: Simplified view of the SRB middleware

The SRB servers process all of the client requests. To access an SRB server, the client must use the APIs (high-level, low-level) offered by the SRB server. The two different APIs are discussed in detail in section 3.8. A high-level request implies the automatic MCAT update through the SRB server while a low-level request implies a manual MCAT update through the client.

If a high-level request arrives at the server, it has to do all the necessary update operations with the zones MCAT server through the MCAT API. If a low-level request arrives, the client has to do these update operations all by itself.

Each zones MCAT server manages all the metadata belonging to the file objects stored in the SRB grid. Metadata include information about the stored data sets, grid user information and information about the storage resources managed by the various SRB servers.

SRB is packaged in only one single package, no extra package is necessary for the MCAT server. While configuring SRB the administrator can enable the MCAT functions within the SRB package, resulting in a special SRB server (*mcat-enabled*) that is compiled with other options. In this thesis, the *mcat-enabled* SRB servers will be called *MCAT servers*.

SRB clients are user applications that access the SRB datagrid through the SRB server API. Special optional features of the SRB server (for example GSI-auth, SSL, etc.) require additional software from the client. The client has to know the compile options of the SRB server he wants to connect to before he can connect to the SRB server.

Starting up SRB brings up one main daemon process, the *SRB master*. The *SRB master* listens for incoming connections. If a connection request comes in, the SRB master process spawns one worker thread, the *SRB agent* process. The *SRB agent* process is described in detail in the subsequent paragraph.

3.1.2.3 SRB Agent

When a user request is received by the *SRB master* process, it forks an *SRB agent* process. From that moment, the complete communication between server and client is done by the *SRB agent* process.

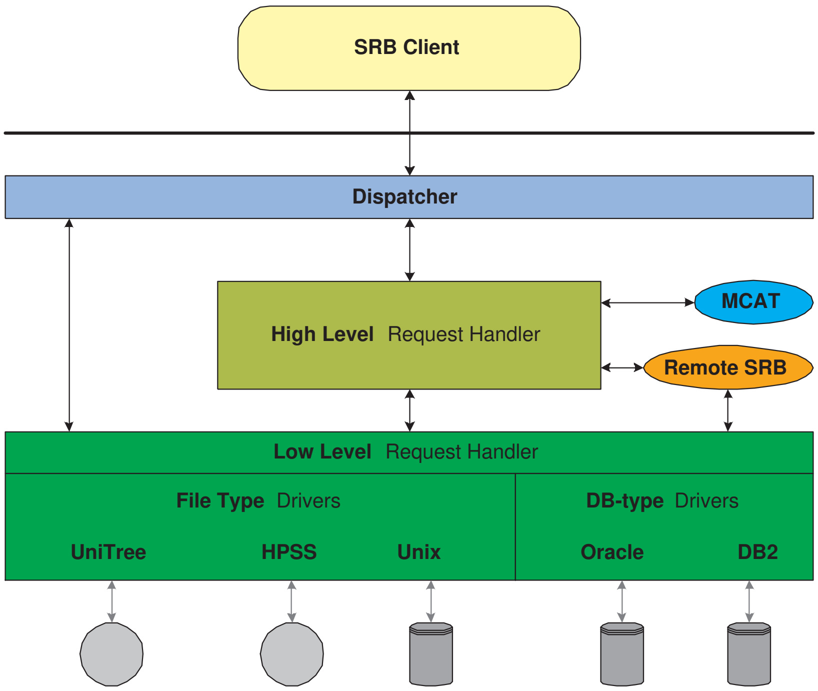

Figure 3.2: SRB Agent

Figure 3.2 depicts the schematic structure of an *SRB agent* process. An *SRB agent* process consists of three main components, *Dispatcher*, *High-Level Request Handler* and *Low-Level Request Handler*.

After establishing a connection between *SRB master* process and client, the *SRB master* forks an *SRB agent* process and passes it the User-ID and the dataset name from the client request.

After forking, the *SRB agent* process makes an MCAT Call to validate the UID and dataset name. If the authentication data is correct, the agent processes the clients request.

The *Dispatcher* is located at top of the SRB agent process. It receives the client requests and dispatches the client requests either to the *High-Level Request Handler* or to the *Low-Level request Handler* whatever kind of request was received.

If the transmitted client request is a low-level request for the local SRB agent, the request is passed to the *Low-Level Request Handler* and the storage access is done by the handler. Is the received request to be processed by a remote SRB agent (the requested files are located at the remote agent), the request is piped to the appropriate SRB agent. Is the request successfully processed, the low-level process handler gives the results back to the dispatcher. The dispatcher sends the result back to the requesting client or to the requesting remote SRB agent process. In case of a low-level request, the *SRB agent* makes **no** update request to the MCAT server. All the necessary MCAT updates have to be done by the requesting client itself.

In case of a high-level request, the SRB agent updates all the necessary metadata in the MCAT server. If a high-level request arrives at the dispatcher, it dispatches the request to the *High-Level Request Handler*. The high-level request handler processes all the metadata queries and manipulations necessary for completing the request. In case of a local processable request, the high-level request handler constructs several low-level requests that are being directly sent to the low-level request handler. If the request has to be processed by a remote SRB agent, the high-level request handler bypasses this request to the corresponding remote SRB node. After successfull request completion, the low-level request handler (local or remote) sends the results to the corresponding high-level request handler. If the metadata updates are all successfully done, the high-level request handler can return the completed request to the dispatcher. The dispatcher then returns the results to the client.

3.2 Namespace

Within a datagrid, file objects are stored on different, heterogeneous media. Storage media include local hard disks, *Storage Area Networks (SANs)* or tape-silos. To give the user uniform access to its data, the datagrid software generates a new namespace. Within this namespace, uniform access to all files should be realized. This namespace has to be a grid-global namespace for the datagrid to allow uniform access. Uniform access means that all files are accessed equally, regardless of their physical storage location.

To realize a grid-global namespace the datagrid has to fulfill the following two requirements:

- Uniform API for all users and file-objects
- Metadata management for all file-objects. Important metadata regarding the global namespace are the physical location of a file-object, the logical name visible to the user and a globally unique file-object ID.

To manage the metadata in an efficient way, both analyzed datagrid software store their metadata in a database. This metadata-database is responsible for generating of the grid-global uniform namespace.

Within the namespace, both software implementation generate a file-hierarchy which is comparable with the file system tree of a modern file system (this is called *directory-like*). Several file-objects are aggregated in a directory. This directory has exactly one superior directory (apart from the top-level directory) and perhaps several sub-directories. With the help of the described steps, a structure similar to a directory-tree is generated.

The subsequent paragraphs describe the global namespaces for the analyzed software and at the end of this section, the namespaces will be compared against each other.

3.2.1 European DataGrid

The EDG organizes its global namespace in so called *Virtual Organizations (VOs)*. A *Virtual Organization* is an abstract grouping of datagrid users to subdivide the global namespace. No interaction between the different *VOs* is planned, so every *VO* has its own local namespace.

The files stored in EDG are addressed through URLs which contain the servername where the file is stored, the name of the *VO*, and the file name on the server.

As EDG uses the *LDAP [28]* directory service, the global namespace is organized in a directory-like structure.

3.2.2 SRB

SRB organizes the global namespace into so called *zones*. A zone is comparable to the VO abstraction that is used by the EDG. Inter-zone transfers are possible, but not the usual case. Within each zone, the SRB namespace is to be seen as a closed unit.

The metadata of all file-objects in SRB are managed by the MCAT server. Supported file-object types by SRB are files on filesystems, data sets, database table-objects and database large-objects. Supported *Database Management Systems (DBMS)* are Oracle, DB2, PostgreSQL and Sybase.

The MCAT service is contacted by the *SRB agents* via an API, which consists of all necessary functions to query and manipulate metadata. Within the MCAT service, file-objects are organized as a tree of *collections*. Each *collection* is an aggregation of several file-objects. A *collection* has exactly one parent collection (apart from the top-level collection) and perhaps sub-collections. This construction spans a *collection-tree* which is comparable to the directory-structure of a file system.

All file-objects have a globally unique logical name. This logical name is being mapped to a physical filename by the MCAT server. Compared to the directory structure of a filesystem the logical name in SRB does not contain implicit information about the physical storage location of the file-object. A collection consists of file-objects from different physical locations to abstract the physical storage location completely from the logical name.

Affected by this strong abstraction, the user does not need any information about the physical storage resource. The user is able to work solely with the logical name provided by the SRB system.

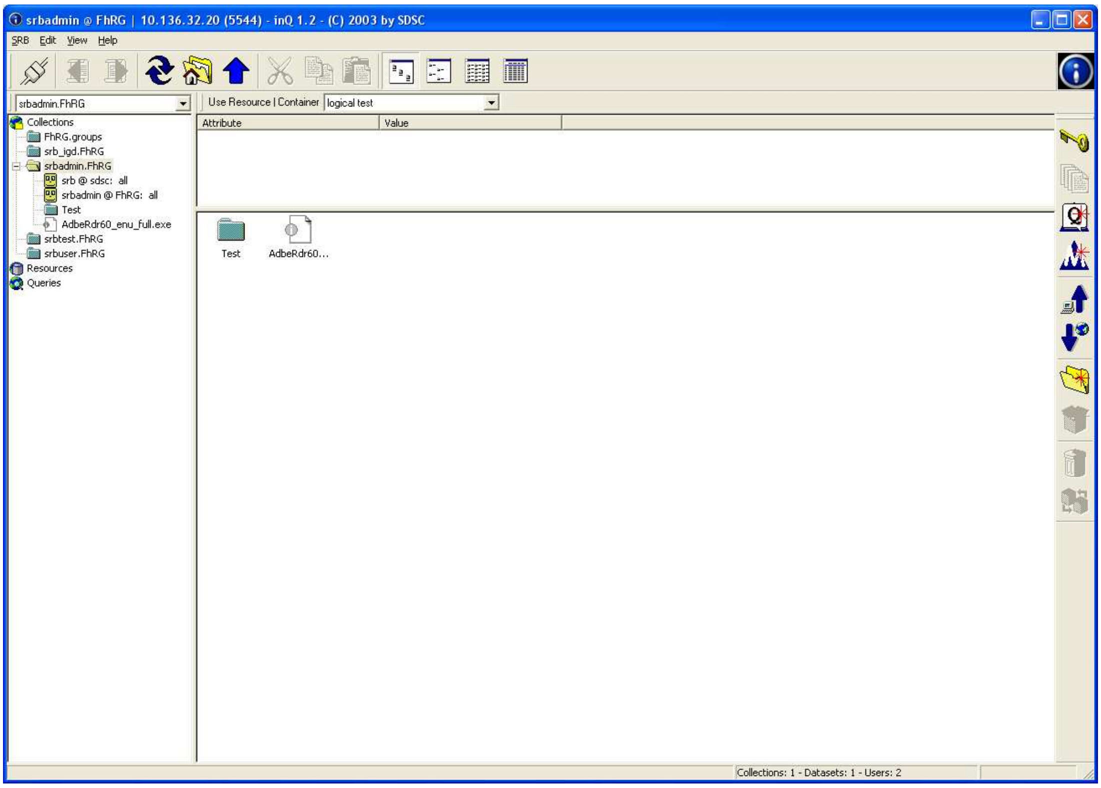

Figure 3.3: InQ SRB Client

Figure 3.3 shows the windows-based client application *InQ*. The left window shows all the collections managed by the SRB system. The main window shows the contents of the currently selected collection. The top-window is responsible only for the metadata management. Right to the main window and on the top of the windows are task bars that contain shortcut icons for all commonly used *InQ* function.

3.2.3 Summary

Both EDG and SRB generate a homogeneous namespace that can be accessed in a uniform way. They span up a directory-like structure within which the data-grid users can navigate through their data structures. No significant difference between the different realizations of a homogeneous, uniform namespace can be pointed out.

3.3 Storage Media Abstraction

Within a datagrid, many file-objects of different sizes are stored on diverse storage hardware. The management of all these different file-objects and storage hardware is the challenge of the datagrid management software.

Caused by large file-size differences, the file-object I/O performance is very poor. For example, at worst case, each writing operation needs an overhead-time for inserting, rewinding, seeking and ejecting the tape in the archival storage system. Through different optimization mechanisms, the file-object I/O can be improved a lot.

Many different storage systems of different sizes can be connected to a datagrid storage server. Management of these different storage hardware components is an additional challenge to every datagrid management software. The analyzed software meet this challenge different.

The different optimization mechanisms, the storage hardware management and the supported media types of the analyzed software are discussed in this paragraph.

3.3.1 European DataGrid

The EDG has to be seen as a closed storage system. Thus, all files have to reside on EDG *Storage Elements (SEs)* before the datagrid users are able to access the files. EDG does not support any mechanisms to optimize file storage yet.

All the files managed by EDG reside on an SE that is assigned to the EDG system. Metadata containing essential information about the files is located on a *Replica Catalog (RC)*. Each *VO* (section 3.2) has its own RC to store the file and user metadata.

The EDG file replication tool *GDMP* (section 3.7) offers an interface to access a *Mass Storage System (MSS)*. This interface is not to be seen as an *MSS driver*. The interface is only a plug-in mechanism for staging scripts since different sites have different MSSs. *Hierarchical Storage Manager (HSM)* systems are supported as well. Through the HRM API, many MSS systems can be accessed. The HRM support is realized as a plug-in to the native MSS interface of the GDMP.

3.3.2 SRB

The reading or writing of many small files causes a large overhead at archival storage systems. Caused by this overhead, the I/O performance for small files on large archival storage systems is very poor. To avoid this problem, SRB invented the *container* abstraction.

Many small files can be aggregated in cache to a *container* object. The resulting container object can be written as a large file-object to the archival storage system, whereby the file-object I/O performance on the archival storage system is being improved.

A little example calculation:

Let:

n	number of small files,
T_{oh}	the overhead for inserting, rewinding, seeking and ejecting,
T_{rw}	the pure reading/writing time,
T_{all}	the overall time

Standard case (without container):

$$T_{all} = n * T_{oh} + n * T_{rw}$$

Overall time with containers:

$$T_{all} = T_{oh} + n * T_{rw}$$

This little example calculation shows the saving of time and the corresponding performance boost of the container mechanism. This calculation is to be seen as very simplified. The overhead-time is assumed constant and the time to pack the containers was neglected (because this can be seen as a real-time process).

To make the file-object stored in a container accessible by the user, MCAT and the I/O drivers had to be adapted to support the container feature. The MCAT database has now to store additional meta-information regarding the container-objects.

If a container-inside file is to be accessed, the complete container is loaded from the permanent or archival storage to the caching instance. After extraction of the container in the cache, the requested file is accessible for reading and writing. All modifications on the file are done in the cache exclusively. If a *sync* request arrives or the container is about to be thrown out of the cache, all container-files are packed together again to a single container and written back. Section 3.4 discusses all the details concerning the caching of containers.

Physical storage resources are described in MCAT exactly through some meta-information. This meta-information includes resource-location (hostname and path), media-type (UNIX filesystem, HPSS filesystem, DB table-object, DB large-objects) and resource-type (*cache* or *permanent*, *primary* or *normal*). File-objects located on *cache* resources can be substituted, file-objects stored on *permanent* or *archival* resources can not be substituted. The substitution is the deleting mechanism with the caching mechanism of the datagrid.

At creation time of a container-object, it is assigned to a *logical resource*. *Logical resources* aggregate one or more physical storage resource that are registered with the MCAT server. The aggregation of one or more physical resources can either be used to enlarge the storage space or to create a redundant resource. Each logical resource is able to aggregate more than one physical resource of one specific resource-type (*cache*, *permanent* or *archival*). If a logical resource aggregates more than one physical resource of one resource-type, there has to be exactly one physical resource marked as *primary*.

As the complete container management is made in the cache area of the SRB server and the container must write back to the permanent or archival storage, the logical resources used to create containers must contain physical resources of both object-types (*cache* and *permanent/archival*).

3.3.3 Summary

Storage Media Abstraction is the capability of a datagrid implementation to abstract the real storage hardware structure in the backend. Thus, the whole storage hardware appears as one or more homogeneous storage pools that can be accessed in a uniform way.

Both analyzed datagrid software support a wide spectrum of storage hardware in the backend. There are no significant differences between the different software.

SRB supports an optimization algorithm both to speed up the file-access on slow storage hardware and to maximize the degree of efficiency of the storage media.

3.4 Caching

To use heterogeneous storage media in an effective way, introducting a caching structure within a datagrid storage node seems to be reasonable. Through the use of faster storage media (for example local harddisks) as caching instances and the usage of the slower storage media (for example tape-silos) as archival or permanent components, the I/O performance can be speeded up.

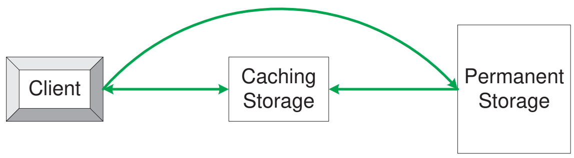

Figure 3.4: Write-Through Caching

The usage of cache instances requires a complex caching-management, comparable to the caching-management in the caching hierarchy of a desktop PC. In general, the caching instances are of smaller size than the archival or permanent instances. For that reason, a caching replacement strategy has to be designed. If the caching instances are out-of-space and data wants to be loaded in the cache, any of the other cache content has to thrown out of the cache.

The caching instance works as fast memory instance, whose contents can be modified. To synchronize the *modified (dirty)* contents, the two main strategies *write-through* and *write-back* are possible.

If the *write-through* strategy is used, all modifications on a data content are written in the cache-copy of the content and to the permanent-copy of the content additionally. On the one hand, this strategy is a slow strategy but on the other hand, data inconsistencies can not appear.

The *write-back* strategy does not generally synchronize the cache with the permanent storage. A synchronization is done, if the 'dirty' content is to be thrown out of the cache. This strategy is the faster one, but data inconsistencies can appear, if the synchronizing fails.

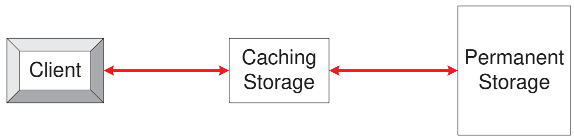

Figure 3.5: Write-Back Caching

3.4.1 European DataGrid

The current release of EDG does not support a hierarchical caching as described above. The EDG user needs to do manual caching by moving files from a archival storage system to a fast caching system. The whole cache substitution is to be done at the instigation of the EDG user.

3.4.2 SRB

In section 3.3, the logical resources abstraction of SRB is explained. A logical resource aggregates one or more physical resources. Physical resources that are contained in a logical resource are marked either as *cache*, *permanent*, or as *archival*.

Physical resources marked as *permanent* or *archival* are used for permanent file storage. Caching-algorithms do not affect the contents of these types of storage. SRB uses physical resources marked as *cache* for its *write-back* caching algorithm. The according physical resources marked as *cache* within a logical resource are used as caching instance only inside the corresponding logical resource. SRB does not support global caching.

If a logical volume aggregates more than one physical resource of one particular type, there has to be exactly one physical resource marked as *primary* of this type. Thus, if at least two physical *cache* resources exist, one of them has to be the *primary* cache. Further information can be obtained from section 3.3.

If no caching resources are defined within a logical resource, all data modification is written directly to the *permanent* or *archival* resources. If a logical volume does not contain at least one *cache* resource, the use of *containers* (as described in 3.3) is forbidden. This special case will not be treated any further within this section, it is assumed, that there is at least one *cache* resource within the logical volume.

After receiving a file access request, the SRB searches in the caching resources for the file. If the file is found, the request is processed. Otherwise, if the file is not localized in the cache, the SRB server loads the requested data from the permanent storage resources into the cache. Loading of new data into the cache might require substituting operations, thus a cache substitution might be necessary.

At modification time of a data content, the modification is only done in the cache (*write-back* strategy). The modified data content is only written back on a cache-substitution or an explicit *sync* command. On writing back the modified data, the data content marked as *dirty* is written back to its original location.

As mentioned above, a logical resource has to contain exactly one *primary* resource per type. If a custom I/O operation occurs, the *primary* resource is being favored. In case of an I/O request, the SRB server first searches in the *primary* cache for the file. If the file is not located in the *primary* resource, the next resource is asked for the file.

3.4.3 Summary

The analyzed datagrid software implement the task of hierarchical caching in different ways. The realization of the caching depends strongly on the designated approach of the respective software.

The EDG does not support a hierarchical caching structure as in the SRB implementation. As most of the data structure are huge datasets that can be downloaded from one master server through EDG and most of the data is read-only, the EDG does not really need a hierarchical caching structure.

SRB supports a hierarchical caching structure that supports automatic data synchronization. As the cache has to be assigned to one designated logical volume, on the one hand, the caching space is a very static storage space that can only be used in one logical resource. On the other hand, the strict differentiation between the various caching areas, data integrity is maximized, applying of strict security concepts is made easy, and due to the clear structure, the affinity to generate errors is minimized.

For use in the FhRG context, the SRB caching solution seems to be more adequate, as the EDG does no caching which is required to speed up the working procedures.

3.5 Replica Management

One central challenge of a datagrid is the creation and management of data-replica. Through the use of data-replica, the data access-time is optimized.

As all replica of a file-object can be modified, synchronizing problems may occur. Some synchronization problems are listed below:

- When should be synchronized?
- Which replica are preferred by synchronization operation?
- What part of the dataset has to be synchronized? Or has the whole dataset to be synchronized?

The analyzed software both meet the challenge of data-replication in a different manner. This section analyzes the realization of data-replication in the two analyzed software packages. At the end of the section, these realizations are compared against each other.

3.5.1 European DataGrid

The current version of EDG supports only the manual data-replication. If a user wants to create one or more replica of a file-object, he has to use one of the functions *gdmp_replicate_get ()* or *edg-replica-manager-replicateFile ()*.

An automatic data-replication is planned to be implemented in the next release of EDG.

As only manual replication is possible, no synchronization mechanism is implemented, yet. The user has to synchronize the replica of a file-object manually.

3.5.2 SRB

SRB supports both automatic and manual data replication.

The automatic replication mechanism generates data-replica at file-object generation-time automatically. To use the automatic replication mechanism, the client has to select a logical storage resource (see section 3.3) that contains more than one physical resource marked as *permanent* or *archival*. Additionally, the user has to set a flag (COPIES=ALL) within the file-object metadata. If these requirements are fulfilled completely, the SRB server generates file-object replicas on all physical resources of the logical volume the file-object is stored on.

To create a file-object replica by hand, the user has to call the function *srbObjReplicate()*. With this function, the user is able to create a file-object replica everywhere in the datagrid.

3.5.3 Summary

As EDG does not support automatic file replication, the EDG may not be applicable in the FhRG environment.

The SRB supports both the manual and the automatic data replication. To execute the manual replication feature, datagrid users tell the system to replicate the corresponding data to another location in the datagrid. The automatic data replication can be configured by datagrid users to replicate specific data automatically.

3.6 User Management

To manage a datagrid in a multi-user environment, it is necessary to authenticate each user and authorize this user according to the specified permission scheme.

User-management is one essential point in both analyzed software. The particular implementation of the user-management task differs in various aspects.

In this section, the particular user-management implementations are being presented and at the end, they are being compared against each other.

3.6.1 European DataGrid

EDG supports the *Grid Security Infrastructure (GSI)* authentication scheme. GSI is developed by the Globus Group [29] intended for use in their *Globus Toolkit (gtk)* software. It is based on *public key encryption* [30], *X.509 certificates* [11] and the *Transport Layer Security (TLS)* protocol (formerly known as *SSL*[12]). GSI is discussed in detail in section 4.1. As GSI supports the *single sign-on* feature, authenticated datagrid users only need to log-in once to the whole system. No additional login to a system node is required.

EDG organizes its users in so called *Virtual Organizations (VOs).* On a task request by a datagrid user, the users certificate is compared with a file that is populated by information from the VOs. Each VO has its own working environment. VO members can use the resources of the VO, users that do not belong to the VO are not allowed to use this VOs resources.

To login to the EDG, a machine with an EDG user-interface installed has to be accessible by users. Access is controlled via a *certificate-proxy.* A certificate-proxy is a limited-lifetime credential that is signed by the users private key. Subsequently, all authentication is done by the certificate-proxy. Certificate-proxies are essential for the GSI concept, that is described in detail in section 4.1.

3.6.2 SRB

SRB organizes its users in so called *zones.* A zone is comparable with the VO abstraction of the EDG software. Zones give the SRB administrators the ability to partition users according to their needs. Inter-zone logins are supported only between trusted zones. In the further analysis of the SRB user-management capabilities it is assumed that all users reside in a default zone.

The SRB supports six authentication schemes:

- **Grid Security Infrastructure:**
  Grid Security Infrastructure (GSI) is a software package developed by the Globus Group as a component in their *Globus Toolkit (gtk).* The GSI is discussed in detail in section 4.1.

- **Encrypted GSI:**
  Likewise the GSI authentication scheme with an encrypted submission of the user credentials.

- **SDSC Encryption / Authentication:**
  The *SDSC Encryption / Authentication (SEA)* authentication scheme was developed by the SDSC as itself. As SDSC stopped development of the SEA authentication scheme, this product is about to be discontinued very soon. SEA will not be considered anymore in this thesis.

- **Encrypted SEA:**
  Likewise the SEA authentication scheme with an encrypted submission of the user credentials.

- **Plain Password:**
  This is the classical password authentication scheme without password encryption. To authenticate, a user has to enter his username and password. If the username/password pair is valid, the specified access permissions for this user are applied.

- **Encrypted Password:**
  Likewise the *plain password* authentication scheme with an encrypted password submission.

Each file-object stored in SRB has a metadata entry in the MCAT server. One of the meta-information of this metadata entry is the *Access Control List (ACL)*. This file-specific ACL consists of one permissions list for each registered user.

The permissions entry is realized as a link to a list of permissions. There are predefined template lists of permissions that cover nearly all types of commonly used permission schemes. The system administrator is able to create new customized permission lists, that can be linked from an ACL entry.
An ACL entry is a triplet:
$< dataObjectId, userId, PermissionId >$
where:
dataObjectId is the system-internal identity of the corresponding file-object, the *userId* entry represents the user-identity. *PermissionId* gives the identification of the permissions list.

SRB offers two ways to administer datagrid users. User-management can either be done via some API calls or via *mcatAdmin*, the SRB/MCAT administration graphical frontend. Only SRB users with administrative access are allowed to do user administration tasks.

Tickets

The *Ticket* mechanism is implemented in SRB to enable registered users of the datagrid to share their data with other registered or unregistered users of the datagrid. This *Ticket* mechanism works as follows:

1. The file-object owner is able to grant read-only access to his file-object. To grant the read-only right to another user, the file-object owner issues a ticket to this user.

2. A ticket is a 10-digit alphanumerical string that is transmitted from the ticket-issuer to one or more ticket-users. After receiving a ticket from another user, a user is able to read the file-object specified in the ticket of the ticket-issuer.

3. Tickets can be issued to both MCAT-registered and MCAT-unregistered persons.

   If a unregistered person wants to use a received ticket, it has to execute a special API call to contact the SRB server. The normal user-authentication is bypassed and the connection between unauthorized user and SRB server is in a non-privileged state.

3.6.3 Summary

The user management capabilities of the software are analyzed in this section. All analyzed software can be operated in a multi-user environment.

The EDG organizes its users in so called *Virtual Organizations (VOs)*. Each VO owns its specific security space and namespace. Thus, a VO manages only its own members. No inter-VO authorization is implemented. A member of a VO can not access data belonging to another VO.

This authorization mechanism enables the administrator to differentiate strictly between different user partitions. As each user has to be a member of exactly one VO, the user is caught in its own namespace. This is a good approach for the intended usage of EDG where many organizations access the datasets. The authorization mechanism satisfies all the FhRG requirements. It uses a certificate-based authentication scheme, thus, the EDG authorization mechanism would fit in the FhRG authentication scheme.

SRB organizes its users in so called *zones*. A zone is comparable to the VO abstraction of the EDG software described above. Within the zones, users are organized in so called *domains*. Within domains, user groups can be arranged, thus, a fine distinction between different types of users can be made by the SRB administrators. SRB is capable of the GSI-based authentication scheme which is the authentication scheme used in the FhRG. GSI is explained in detail in section 4.1.

Respecting to the user authentication, there is no significant difference between the software. They both support the GSI authentication scheme and a good hierarchical user management capability.

3.7 File-Transfer

This section outlines the software file-transfer mechanisms of the analyzed software. A file-transfer might be a transfer of data between datagrid client and server or a data-transfer between miscellaneous datagrid servers.

Through uploading and downloading files to or from a datagrid server, its clients are able to use the datagrid resources. File-synchronization and data-replication is done by the datagrid storage servers through a direct connection between these servers.

GridFTP [15] is a popular file-transfer protocol used by many storage systems (DPSS, HPSS), distributed filesystems (DFS), datagrid software and many more. It combines portions of RFC 959 (*File Transfer Protocol (FTP)* [31]), RFC 2228 (*FTP Security Extensions* [32]), RFC 2389 (*Feature negotiation mechanism for the File Transfer Protocol* [33]), and *IETF*[1] draft *FTP extensions* [34]. Through this combination of features, *GridFTP* includes the following capabilities:

- *GSI* and *Kerberos* [35] support
- Third-party control of data transfer:
  One GridFTP instance on computer A is able to transfer files from computer B to computer C without streaming the data through computer A.
- Parallel data transfer:
  Splitting of files on one single data server to transfer data over multiple data connections.
- Striped data transfer:
  Distribution of data transfers on multiple data servers over multiple connections.
- Partial data transfers.
- Automatic negotiation of TCP buffer/window sizes.
- Integrated instrumentation.

As inter-organizational file-transfers are relatively slow, the file-transfer speed is the main performance bottleneck of each datagrid. With respect to these performance issues, each analyzed datagrid software handles the file-transfer in a different way.

Additionally, some software offer data-transfer encryption and/or compression to fulfill required security issues.

The data-transfer mechanism realization of the two analyzed datagrid software are discussed in this section. At the end, the analyzed software are being compared against each other with respect to the file-transfer mechanism.

[1]IETF $\hat{=}$ Internet Engineering Task Force *http://www.ietf.org*

3.7.1 European DataGrid

The EDG software offers three different ways to handle its data-transfers. The offered tools have different intentions of what kind of data-transfer they are about to handle. These three tools are shown below.

EDG uses the *Grid Data Mirroring Package (GDMP)* [36] to manage all its data-transfer tasks within the EDG. The GDMP is a collaboration between the EDG project and the *Particle Physics Data Grid (PPDG)* [4] project. GDMP version 4.0 is part of the official EDG software system and contains some modifications for the EDG testbed.

Another file-transfer tool with limited functionality is the *EDG Replica Manager (ERM)*. The ERM is only able to replicate single files, has no mass storage interface, but it is able to transfer files between EDG Computing Elements and EDG Storage Elements within a EDG datagrid.

To upload or download a file to or from a server, the client uses the *globus-url-copy* tool. This tool is capable of third-party transfers, thus, a client is enabled to copy data from one remote node to another remote node without piping the data through the local machine. The *globus-url-copy* tool uses the GridFTP protocol to transfer data.

Grid Data Mirroring Package

The *Grid Data Mirroring Package (GDMP)* offers a set of C++ API functions and command line tools for all supported file replication operations. It is used for data-transfers between different *Storage Elements (SEs)* of an EDG datagrid. GDMP maintains a local file catalog for the specific SE it runs on. File-transfers in GDMP are done by the GridFTP protocol.

To manage the file-replicas, the GDMP offers a set of command line tools and a C++ API to a generic Replica Catalogue. An additional C++ API is offered by GDMP to handle information from EDG Scheduling systems.

The GDMP is able to replicate sets of files between *Storage Elements (SEs)* of EDG. It has a mass storage interface to address various mass storage devices. File-transfers done with GDMP are reliable as a CRC check proves the file integrity after successful transfers.

3.7.2 SRB

SRB offers two main functions (*Sput()/Sget()*)to upload/download files to and from the datagrid server in addition to the respective file-transfer API functions (a detailed API description is given in section 3.8). *Ssyncd()* and different *sync()* API functions handle the synchronization tasks between the different SRB servers. Figure 3.6 depicts the SRB file-transfer scheme.

SRB uses the *Unix© Socket* architecture to handle all its connections.

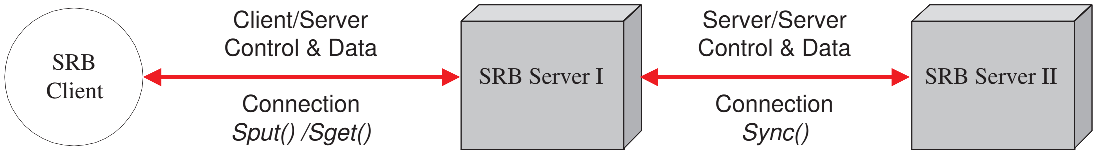

Figure 3.6: SRB File-Transfer Scheme

If an SRB storage server receives a file-transfer request, it opens a new TCP socket which is about to handle the file-transfer. The assigned port is selected by picking one port of the port-range that is given to the SRB at compile time. Compile options of the SRB software are given in Appendix A.3 and A.4. After finishing the file-transfer this newly generated socket is closed.

No parallel transfer of one file or partial file-transfers are possible through the generic use of sockets. As the SRB manages a set of ports, parallel transfers of multiple files are possible.

As of release 2.1.0, SRB supports the encryption and compression on network traffic. A GSI-based encryption was planned, but the developers of SRB decided to create an own encryption/compression scheme, which provides more security, better performance, and compression as well as encryption.

A GridFTP driver is planned, but not yet implemented. The GridFTP support will come up in one of the next major releases.

3.7.3 Summary

The section File-Transfer describes the basic mechanisms of transferring files from one datagrid node to another.

EDG uses the GridFTP protocol developed by the Globus Group. This protocol is a fast, secure and reliable file-transfer protocol. The EDG implementation of the file-transfer task fulfills all requirements of the FhRG environment.

The SRB uses its own socket-based protocol that reduces network overhead, but has limited capabilities compared to the GridFTP protocol. The socket-based implementation of the file-transfer task within the SRB implementation misses some features for a fast file-transfer. However, with the GridFTP support to come in the next releases, this problem can be tolerated.

3.8 API

By offering an *Application Programmers Interface (API)*, a datagrid service provider gives its clients the ability to use the datagrid services. Both analyzed software offer at least one API.

This paragraph contains a short description of both API realizations. At the end of the paragraph, the APIs of the analyzed software are compared against each other.

3.8.1 European DataGrid

European DataGrid (EDG) offers a Java API, a C++ API and a *Command Line Interface (CLI)* to allow its users access to the datagrid services. The Java API and the CLI contain the same commands, thus, they are analyzed together. The C++ API consists only of three functions, that are not very useful. Thus, the C++ API will not be analyzed within this thesis.

The Java API/CLI can be split up into four separate paths:

1. **Management API:**
   Through this high-level API, management of files combined with the necessary metadata-update operations can be made. It contains functions to push locally stored files into the EDG, to create replica of files and to delete files from the EDG system.

2. **Database API:**
   This low-level API manages all the update operations concerning metadata within the metadata system. These updates are necessary after execution of functions contained in the *Optimization API* or the *Filetransfer API*. It consists of functions to register or unregister files or GUIDs within the EDG, to gather information about replicas or GUIDs and one function to set a new GUID alias.

3. **Optimization API:**
   This API intends to optimize the file-transfers within the EDG datagrid. It is to be seen as very experimental. The optimization API consists of a *list()* and a *get()* function to optimize the costs of file-transfers.

4. **Filetransfer API:**
   Commands contained in the file-transfer API are responsible for file-transfer operations only. After command execution, no metadata update is done. The API consists of only one function to copy files from one place to another.

3.8.2 SRB

SRB offers an API to process the communication between SRB servers and clients. Main functions of the SRB API can be split up into a *high-level* and a *low-level* section.

The *high-level* functions are *complete* functions, which means they complete both the requested operation and the necessary metadata update operations.

If *low-level* functions are used by the client, the requested operation is processed, but *no* metadata updates are done by this functions.

Both, a C++ and Java API are offered by the SRB datagrid software. The C++-API is a fully-functional API that supports all operations on an SRB server. The Java API is still under development. Not all functions necessary to manage an SRB server are implemented yet. As the Java API implements the same functionality as the C++-API, they are analyzed together.
The whole API can be clustered in the following parts:

1. **Client/Server Connection API:**
   This API is used by the client to manage its connections to the SRB server. It contains functions for connecting and disconnecting to an SRB server and one function to query the error code of the current system call.

2. **High-Level API:**
   As mentioned above, the high-level API is responsible for the complete data management. After a completed request, the necessary metadata update operations within the MCAT service are done.

   It consists of all necessary functions for dataset management, data collection management, data-container management, ticket management and file management.

3. **Low-Level API:** The low-level API is used for dataset management without any metadata update operations. After a completed request, each function of the low-level API returns to the caller without doing any update operation in the MCAT server.

   As too many functions are covered by the low-level API, the description of this API is clustered in three parts according to the supported function type.

   - **Low-Level File-Type API:**
     Functions of the file-type API are responsible for the management of data and directories stored in the SRB datagrid system.

     It consists of functions to create, delete, open, close, read and write files or directories in the SRB system. Additionally, functions for synchronizing files or file descriptors and a seek function for files or file descriptors are implemented.

   - **Low-Level DB-Type API:**
     SRB supports the use of large-objects within the supported DBMSs. The DB-type API is responsible for the complete management of large-objects within a database.

     It offers all necessary functions to manage large-file objects, like *seek()*, *create()*, *delete()*, *open()*, *close()*, *read()* and *write()*.

- **Low-Level DB table object API:**
  The DB table object API is responsible for management of table-objects within a database.

  It offers all necessary functions to manage table objects, like *seek()*, *create()*, *delete()*, *open()*, *close()*, *read()* and *write()*.

4. **MCAT API:**
   To proceed the necessary updates within the MCAT database after a completed low-level request, SRB offers the MCAT API which is responsible for updating metadata in the MCAT server.

   SRB servers use this API in case of high-level requests by the client to proceed the necessary metadata update.

   Through the MCAT API, SRB site administrators are enabled to manage datagrid users and datagrid resources used by the SRB system.

   It offers all necessary functions to query and set metadata regarding the dataset management, user management and resource management.

3.8.3 Summary

The API is needed by client applications to communicate with their datagrid server node.

EDG offers a fully functional Java API that supports all necessary function to maintain an EDG based datagrid system.

The SRB offers a fully functional C++ API and a Java API that is under development to become a fully-functional API.

Both analyzed software offer adequate solutions to manage their datagrid systems. Thus, no significant difference between the offered APIs of the two analyzed software can be notified.

3.9 Decision

The main task of this chapter is to compare two datagrid software against each other and to select one of the two software that satisfies the requirements of the FhRG, the target release platform.

To find an adequate software that is sufficient for the FhRG, some evaluation criteria are defined and the two software are evaluated with respect to the defined criteria.

The compared software both support many important features like the GSI-based authentication method, at least one fully-functional API, uniform namespaces, and efficient file-transfers. All these features are important features for the usage with the FhRG, the target installation platform of the selected software.

One main difference between the SRB and the EDG datagrid software is the replica management. The SRB offers better implementations of the file-replication and replica-management tasks.

Both concepts fulfill all required functional aspects for integration within the FhRG. No significant difference between the two datagrid software can be pointed out. Thus, non-functional aspects are evaluated to find a decision.

One non-functional aspect is the future plans of the projects. The SRB is currently under heavy development, and no project termination is planned at this time. The EDG project started in 2001 and is determined to be finalized in the end of 2003. It will be continued in the *Enabling Grids for E-science in Europe (EGEE)* project. Both projects have different levels of experience. Wile the EDG is a young project with a small experience value, the SRB is established on many sites worldwide for a long time. Thus, the experience level of the SRB project is higher than the experience level of the EDG project.

With respect to the non-functional aspects, installation of the SRB is preferred. The next chapter describes the concepts for using and installing the SRB within the FhRG.

Chapter 4

Concept of Operation

In this chapter, a working concept for the datagrid architecture within the FhRG is developed, and the concepts of distributing the software, software integration procedure, and connectivity solutions of the SRB within the FhRG are described.

The first section shows up fundamental concepts of the *Grid Security Infrastructure (GSI)* authentication mechanism. This mechanism is supported by all major datagrid implementations.

In the next section, a concept of how to build a datagrid is being developed. This concept does not focus on technical considerations, it postulates some general requirements that represent an optimized datagrid architecture.

The concept of the distribution procedure contains considerations on the software distribution within the FhRG to all nodes with respect to the global and local environments.

Subsequently, a concept of the necessary integration procedure is developed. The integration procedure consists of a software embedding within existing structures with respect to the global and local environments. This section contains the development of a firewall concept for the distribution of the SRB software.

As the SRB software has to interact with the existing FhRG software, considerations about the connectivity task must take place. This includes a high-level description of the API, the ability of the API to interact with client software, and some considerations about the API capabilities.

Finally, a few use cases with configuration examples are shown. These use cases represent a few common scenarios with the corresponding configuration options.

4.1 Grid Security Infrastructure

To meet the requirements for secure authentication in a grid environment, the *Globus Group* developed the *Grid Security Infrastructure (GSI)* package. GSI is based on *public key encryption ([30] chapter 8)*, *X.509 Certificates ([11]) and*

Transport Layer Security (TLS, [12]).

The descriptions of the various mechanisms in this section are taken from [29]. The central requirements for an authentication mechanism used within a distributed system are:

- The communication between multiple nodes of a distributed network has to be secure. Secure communication signify an authenticated and preferably trustworthy connection between the nodes of communication.
- Secure communication must be possible between different organizations. Thus, a centralized security management system must not be allowed.
- A *single sign-on* mechanism has to be supported. The single sign-on mechanism gives the user the ability to use all distributed resources of the system with a single log-in. Thus, a user only has to log in once with the *single sign-on* mechanism and after successful authentication, it is able to use resources on all nodes of the distributed environment.

The subsequent subsections describe the working mechanisms of GSI. In the first subsection, the concepts of public and private keys is discussed. The second subsection describes the usage of certificates used by GSI. To allow a secure communication, an authenticated connection is required. The concepts for an authenticated connection are presented in subsection 4.1.3. In subsection 4.1.4, the concepts for preferably trustworthy communication is discussed. Considerations regarding the secure storage of private keys are made in this subsection, too. The single sign-on feature requires the usage of so called *certificate-proxies*. The concepts of certificate-proxies are shown in the last subsection of this section (4.1.6).

4.1.1 Public Key Encryption

The concepts of *public key encryption* presented in this paragraph are taken from [30], chapter 8.

Introduction

Before using the public key encryption, each security instance has to generate a key pair. A key pair consist of a public and a private key of the instance. A security instance is a person or a machine that can be authenticated in a security system.

In the public-key encryption mechanism, public keys are used to encrypt the data. As required by the encryption mechanism, the public keys of two or more communications partners have to be mutually known to each other.

Private keys are used to decrypt encrypted data. The data are encrypted by a known communication partner with the appropriate public key. The private key must be kept secure under all circumstances.

A short example shows the concepts of public key encryption. Two partners, α and β, want to communicate encrypted. The transferred data is called *messages* in this example.

Example Communication
If α wants to communicate encrypted with β, α has to send its public key to β. As the transmitted key holds the information how to encrypt messages designated for α, the initial connection is not encrypted. Now, that β knows the public key of α, it can encrypt all the messages for α with the appropriate public key. If α receives an encrypted message from β that is encrypted with the public key of α, α is able to decrypt the message with its own private key.

As α does not know the public key of β, no encrypted communication from α to β is possible. Once α received the public key from β, α will be able to encrypt messages sent to β, too.

After mutual knowlegde of the public keys, a bidirectional encrypted communication is possible.

Reversible Authentication
A subclass of the encryption functions is *reversible*. To use the public key encryption in a reverse way, the encryption of messages is done by the use of private key and the messages are decrypted by the use of public keys. As a result of this encryption method, everyone who knows the public key of an instance, is able to decrypt the messages originated from this instance. This irrational construction can be useful in some special cases.

The *mutual authentication* mechanism uses the reverse authentication to prove the identity of two communication partners. In mutual authentication, one instance tries to validate the identity of another instance by sending an unencrypted random message instance to the to be validated. This instance encrypts the message with its private key and sends it back. After decryption, the validating instance compares the decrypted message with the original message and if it matches the validating instance trusts the identity of the other instance.

The concept for mutual authentication is discussed in detail at paragraph 4.1.3.

4.1.2 Certificates

Within a GSI environment, each user and service of the distributed environment is identified by a personal certificate. A GSI certificate consists of:

- A subject name identifies the owner of the certificate unambiguously. One single person or one single service can be an owner of a certificate.
- The public key of the certificate owner given by the subject name.

- The identity of the signing *Certification Authority (CA)*, which signed this certificate. Through the certificate signing the CA certifies that the public key and the given identity both belong to the subject name.
- The digital signature of the signing CA.

All communication partners have to trust the signing CAs of their communication partners. As the CA is to be seen as a third party that verifies identities, a trustworthy relation to the communication partners CA is required. Otherwise, a trustworthy communication is not possible. GSI certificates are encoded in *X.509 [11]* format.

4.1.3 Mutual Authentication

Through the mutual authentication, two communication partners are able to prove their identity. As a prerequisite, both partners have to know the public key of the opposing party and each partner has to trust the signing CA of the communication partner.

The authentication protocol used in mutual authentication is the *Transport Layer Security (TLS)* protocol [12]. To trust the CA of the opposing party, both partners need to know the CA's certificate. This CA certificate has to contain the public key of the CA and both partners have to trust the CA certificates origin.

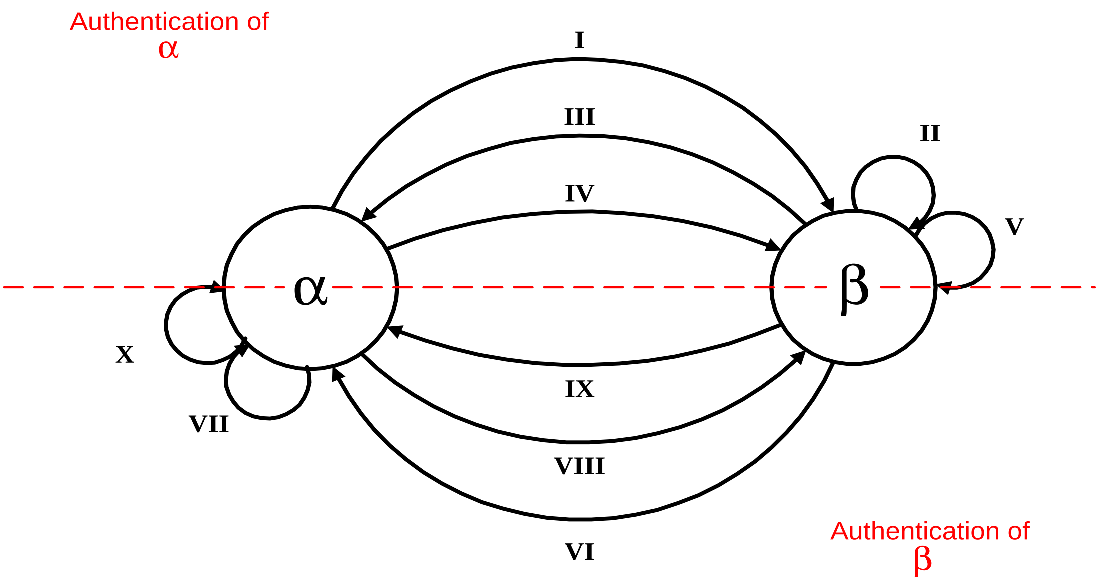

Figure 4.1: Mutual Authentication

Figure 4.1 depicts the process of mutual authentication. The marked actions (**I - X**) are described below:

I The mutual authentication starts with a connection request from α to β. After establishing the connection, α sends his certificate to β.

II After receiving the certificate from α, β extracts the information who α is claiming to be, α's public key and the contained information about the signing CA. Next, α validates the CA signature. This is done by comparing the received CA signature and the already stored (and trusted) CA signature.

III To validate the identity of the certificate's owner, β generates a random message and sends it unencrypted to α.

IV α encrypts the message received from β with its private key. After successful encryption, α sends the encrypted message back to β.

V β decrypts the received message from α with help of the public key from α received in the certificate (step **I**). After successful decryption, β compares the decrypted message with the generated one (in step **III**). If the two messages match, β trusts the identity of α. Now, the complementary procedure has to be done for the validation of β.

VI β sends his certificate to α.

VII α extracts the information from the certificate.

VIII α sends a random message to β.

IX β encrypts the message with its private key and sends the encrypted message back to α.

X α decrypts the message with β's public key and compares the decrypted message with the original message. If they match, α trusts the identity of β.

After completing a mutual authentication successfully, the two partners α and β can trust their identity reciprocal. The whole concept only works if both partners trust the CA of the opposite partner.

4.1.4 Trustworthy Communication

A trustworthy communication can be defined as a secure and confidential communication.

To ensure a secure communication, encryption of the communication paths is necessary. By default, GSI does not encrypt the connection between the communication nodes. After a successful mutual authentication, GSI terminates itself to avoid a connection slow-down. Otherwise, the connection speed will slow down caused by the continuous encryption and decryption of messages. On the other hand, GSI is able to generate an encryption key used for encrypting all subsequent communications between the communication partners.

Another aspect of trustworthy communication is the integrity of communication. Communication integrity means that a third party might be able to listen on a connection, but the sniffer is not able to alter the communication contents. Thus, the contents arriving at one communication partner is the original message transmitted from another trusted communication partner. By default, GSI supports a communication integrity checking algorithm. This feature can be disabled. The overhead caused by this integrity check is to be neglected.

4.1.5 Securing Private Keys

Due to the importance of private keys, they must be stored secure and insular from the access of anyone but the owner. GSI expects the users private key stored on a local file system. To avoid access on the private key, the file in which the private key is stored is encrypted with a password. To use the GSI mechanism, the user has to enter username and password to give GSI the ability to decrypt the private key file of the user.

The GSI programmers are developing a *smartcard* feature. Once this feature is implemented, the user will be able to store his private key on a *smartcard* medium to increase security.

4.1.6 Certificate Proxies

To use more than one resource that requires authentication, a user would have to enter its password for every used resource. This is the standard TLS algorithm, which may be very secure, but not practicable in a grid environment. An Enhancement of the TLS protocol avoids this problem. Through the use of *certificate proxies*, the problem is completely avoided. The usage of certificate proxies lowers the security level a bit, but the level of lowering is quite acceptable compared with the better usability of GSI.

A certificate proxy consists of a newly generated private key and a newly generated certificate. The certificate contains a new generated public key, the signature of the proxy-user and the identity of the proxy-user, slightly modified to indicate that this is a proxy certificate. To avoid misuse, certificate proxies and proxy certificates have limited lifetimes.

The private key of the certificate proxy has to been kept secure, but because of the limited lifetime, the key storage file has not to be encrypted. As the private key storage file is not encrypted, the administrator has to set appropriate access rights to this file.

A modified version of the mutual authentication algorithm is used in the certificate proxy mechanism. The party to be validated not only receives the proxy certificate, but also the user certificate of the proxy user. The public key contained in the user certificate is used to validate the proxy certificate. With this algorithm, an authentication chain is created from the CA through the user to the certificate proxy.

4.2 Global Working Concept

This section develops a conceptual view to the datagrid that will be build and integrated within the FhRG. The introductory subsection gives a short review of the essential points that are important when developing a conceptual view to a datagrid architecture. After the introduction, considerations about a useful datagrid concept are made.

Working examples of the concept developed in this section are given in the section 4.7.

4.2.1 Introduction

A datagrid architecture consists of an arbitrary number of *storage nodes* that span a grid. Figure 4.2 depicts the structure of a datagrid.

All the storage nodes of a datagrid span up a uniform namespace for the datagrid users. Datagrid users are able to access their files in a uniform way, regardless of the node the user is logged on or the physical location of the files.

Each storage node abstracts the own physical resources from the user, thus the datagrid user does not see any physical resources, but only a logical resource that is defined by the datagrid software. This makes it very easy to reconfigure the storage space that is shared with the datagrid.

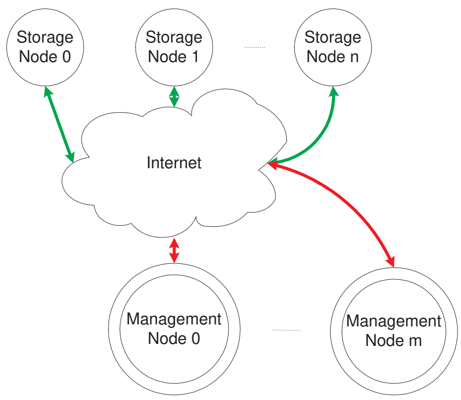

Figure 4.2: Conceptual view

4.2.2 Concept Development

The main focus of the concept development is on the security, availability, fault-tolerance, performance, and ease-of-use aspects of the datagrid that will be built and integrated within the FhRG infrastructure.

For security reasons, an authentication mechanism that is based on the GSI authentication should be used. Additionally, the communication between the storage nodes of the datagrid should be encrypted. In addition, the GSI authentication mechanism is a good choice, as the complete FhRG architecture relies on the GSI authentication mechanism. Another issue of the security aspect is the user account management. Each datagrid storage or management node gets a separate administrative user account. This minimizes the security risk and maximizes the traceability of potential security breachs.

To maximize the availability of the datagrid for all users, every SRB node should accept user logins from all authorized datagrid users. Thus, each user is able to log into each SRB node, regardless if this SRB node is a storage node or the management node. Furthermore, the user access to the datagrid should be granted from all over the world.

From its definition, a datagrid can be characterized as a high-availability software which needs an proper fault-tolerance ithat is stable against network problems. The datagrid nodes are connected through the internet. If a connection black-out occurs for one datagrid node, the system has to react very flexible to this network error. Only the services offered by this node should be absent, all other services should be running as normal as possible. As all SRB services rely on the MCAT service, redundant management nodes running the MCAT service should be used.

To achieve a good performance in the datagrid, each storage node should share a hierarchical storage structure with the datagrid. A hierarchical storage structure implies the use of fast storage at the first level and slower storage elements at the second level. Figure 4.3 depicts the concepts of the hierarchical storage structure.

The fast storage elements can be fast hard disks that act as caches. This means, the data on the fast storage is not meant to be archived data. If the caching storage elements are full, the data is migrated to the larger second level archival storage space. To achieve optimal performance in the datagrid, each SRB storage node should have such a hierachical storage structure.

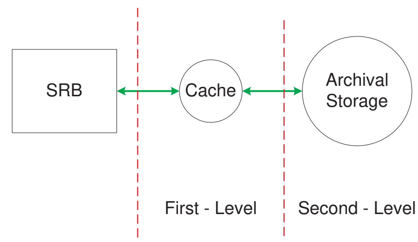

Figure 4.3: Hierarchical Storage

The ease-of-use aspect includes the aspects of easy access to data and easy distribution of data. Easy access to the user data is given by the uniform namespace offered by the SRB. Another point is the easy distribution of data. SRB offers the automatic replication of data. At least one global file storage pool should be created in the datagrid. A global pool is a pool that has some storage space on each datagrid node and the *automatic replication* flag is set. If a user puts data in this pool, the data is mirrored to all available datagrid nodes within the FhRG.

4.3 Distribution Procedure

Currently, the FhRG consists of five institutions at different locations. To use the SRB at full capacity, an SRB installation at every single FhRG location is necessary. Figure 4.4 shows the different locations on a little map. The distribution procedure has to be optimized to minimize both the fault probability and the administration task and to maximize the speed of distribution.

This section describes all the steps necessary to distribute the SRB software. Therefore, the participating FhRG institutes are shown. The presentation is followed by a short description of the software distribution structure.
The five institutes participating in FhRG are:

- **Fraunhofer Institut Arbeitswirtschaft und Organisation (IAO)**, Stuttgart
- **Fraunhofer Institut Rechnerarchitektur und Softwaretechnik (FIRST)**, Berlin
- **Fraunhofer Institut Techno- und Wirtschaftsmathematik (ITWM)**, Kaiserslautern
- **Fraunhofer Institut Graphische Datenverarbeitung (IGD)**, Darmstadt
- **Fraunhofer Institut Sichere Telekoorperation (IST)**, Darmstadt

Figure 4.4: The FhRG Locations
(Source: [13])

First of all, one MCAT-enabled SRB server has to be installed. As the author resides at the ITWM, the MCAT installation will also reside at the ITWM. In general, the MCAT location can be chosen randomly, thus an MCAT installation at the ITWM is chosen.

At least one SRB server is to be installed on all of the five locations. To minimize the risk of errors, installation first takes place in the ITWM. After a successful installation at ITWM, the second distribution step is to install the software at one other site of the FhRG.

Next, the new installation has to be tested. This can be done with a set of configuration options and datasets. If all the tests succeed, the software distribution to all the other sites can be done.

As the ITWM SRB installation is a Linux based installation, the first distributed SRB installation should be a Windows based installation to test the interoperability between the Linux and Windows based installations. If these tests succeed, the distribution structure has reached a high level of modularity. The next installation can be docked to the existing distributed datagrid structure. It does not matter if the docked installation is a Linux or a Windows based installation.

After finishing the SRB distribution, the SRB servers on the different sites span a real datagrid across the FhRG that can be accessed from everywhere around the world. As long as the MCAT server persists, adding or removing SRB servers to or from the datagrid is quite easy.

4.4 Integration Procedure

This section describes the necessary steps to integrate the SRB within the FhRG. Therefore, the global and local environments have to be explained. Subsequently, considerations about the SRB integration within the FhRG are made.

The IAO has three workstations designated as FhRG nodes. The FhRG nodes of the IAO contain the Web Portal service of the FhRG grid services. All the nodes of the IAO reside behind a firewall, so that a *network tunnel* to the IAO nodes has to be established. One of the three IAO FhRG nodes runs the Oracle DBMS, so this node can be used as a LOBJ-based storage system.

The FIRST has a cluster consisting of 16 cluster computing nodes (32 CPUs) and two additional workstations as FhRG nodes. All the FhRG nodes are located within the *DMZ*, so no additional administrative task is required. Section 4.5 discusses the different firewall architectures in detail.

FhRG nodes of the ITWM include a 64 computing nodes cluster (128 CPUs) produced by *Linux Networx [37]*, several high-capacity file servers, and some additional workstations. The FhRG nodes of the ITWM are located within the *DMZ* behind a firewall, thus no further administration tasks are required.

One cluster with 4 computing nodes (8 CPUs) and three additional workstations are the FhRG resources of the IGD. All these resources are located behind a complex firewall structure, so the embedding of the SRB at IGD is a complex task.

The SIT has only a few workstations dedicated to the FhRG. Thus, the SIT does not share any resources with the FhRG at this time.

As an SRB datagrid consists of distributed SRB servers, a firewall concept has to be developed first. The firewall concept has to consider the environmental requirements, security aspects and the datagrid requirements.

4.5 Firewall Concept

An SRB-based datagrid consists of multiple distributed SRB servers at different locations. To integrate an SRB server at one specific location under consideration of the local security concept, a suitable firewall concept for the SRB server has to be developed.

This firewall concept should be a uniform concept that meets all operating requirements of an SRB server.

Operating requirements of an SRB server are the connectivity to the MCAT server, the connectivity to all other SRB servers in the datagrid and accessibility for all datagrid users.
To meet these requirements, the following firewall scenarios are possible:

- **Virtual Private Network**
  ALL SRB nodes are members of a *Virtual Private Network (VPN).* The VPN scenario is discussed in subsection 4.5.1.

- **Proxy**
  SRB makes use of connection proxies as shown in subsection 4.5.2.

- **DMZ**
  The SRB nodes are located in the *DeMilitarized Zone (DMZ).* A detailed description of the DMZ concept and some possible derivates is given in subsection 4.5.3.

A detailed discussion of the scenarios presented above is shown in the three subsequent subsections *VPN, Proxy* and *DMZ.*

The three subsequent subsections *VPN, Proxy* and *DMZ* give a detailed description of the scenarios consequences. After a short operation description, the advantages and disadvantages for each specific scenario are shown.

4.5.1 VPN

The first firewall scenario (the VPN szenario) is shown in figure 4.5.
All the SRB components are located within the *Virtual Private Network (VPN).* The thick red line shows the connection between two physical VPN locations. If the VPN spans more than two locations, a *VPN-Tunnel* between all locations is necessary. With n locations, this yields $\frac{n*(n-1)}{2}$ tunnelled connections.

In this scenario, the SRB nodes are placed within the private networks (behind the firewall) of each institution as shown in figure 4.5. The traffic between the different SRB nodes is *tunnelled* through a VPN-Tunnel. For a detailed description of the VPN concept refer to [38].

This virtual subnet is only for communication between the SRB nodes. Clients can access their local nodes through the private network of their institution.
The advantages of this concept are:

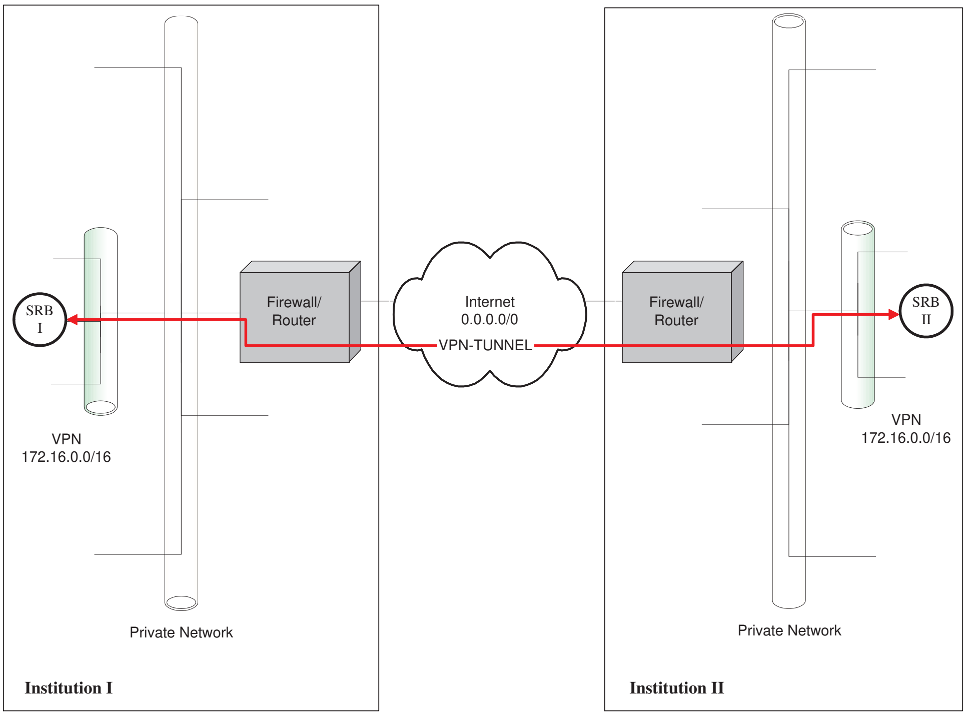

Figure 4.5: SRB over VPN

- Uniform Concept:
  As all installations at the FhRG sites have the same structure, the VPN implementations at each institution can be done in a uniform way.

- Global Namespace:
  As all the SRB nodes are virtually placed in the same subnet, a simple address scheme can be used and no routing problems occur.

- Easy Setup:
  No special networking features have to be considered. A quick-and-easy setup for a small number of nodes is possible.

- Authentication:
  With a central authentication instance, single sign-on is possible within the VPN. The X.509 certificates used for SRB authentication (as described in section 3.6) can be used for VPN authentication.

- Simple Firewall Structure:
  The firewall has only to be open for the VPN-type packets. No additional configuration is needed.

On the other hand, the disadvantages of the VPN concept are:

- Administration:
  With the number of nodes growing, the administration task gets the more and more complex. For each new node, a tunnel from each other node to the new node has to be established.

- Scalability:
  Adding of new nodes within the VPN-based datagrid is a complex task. As mentioned above, the number of tunnels grows with $O(n^2)$, so the scalability of a VPN-based datagrid is very poor.

- Accessibility:
  As the VPN structure is a private and closed structure, *VPN Access Points* have to be created by the VPN administrator. This is necessary to grant users access to the datagrid resources.

To guarantee the accessibility from every client around the world, at least one special SRB node is necessary. This node has to be a member of the SRB VPN and the node has to be accessible from the world.

This concept is a very secure solution. The administration task is too complex for a large number of network nodes. Due to the scalability problem, this firewall solution is not practicable in the required environment.

4.5.2 Proxy

Figure 4.6 depicts the proxy-usage scenario.

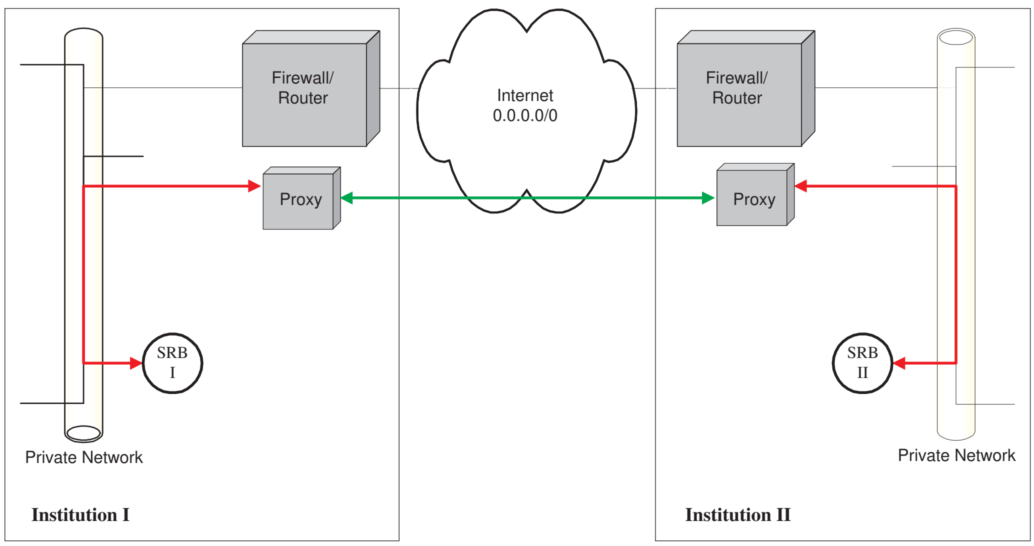

Figure 4.6: SRB Proxy Usage

A proxy instance is added to the networking structure of each institution. All SRB components communicate only with this proxy instance at their local institution indicated by the red line. All inter-organizational communication is done between the proxies, indicated by the green line. Since these inter-organizational connections are not tunnelled, the number of active connections is varying with a maximum of $\frac{n*(n-1)}{2}$ if n proxies are present.

Each SRB node can be integrated in an institute by using a proxy. Proxy usage can bypass the routing instance which enables the SRB to be located in the institutions private network. The SRB node can be accessed from all other SRB nodes and SRB clients through the proxy.
The advantages of proxy-usage are:

- Simple Firewall Structure:
  As only one proxy is used, only traffic from and to this machine has to be allowed in the firewall. No additional firewall configuration is needed.

- Directional Traffic:
  All traffic within the proxy-enabled datagrid is directional traffic from one network endpoint to another network endpoint. Thus, usually, no broadcasts appear on the network and the overhead traffic is minimized.

- Easy Administration:
  The administration task for a proxy structure is very simple. A fully configured proxy can be very easy adapted to new environmental parameters (i.e. IP address change, adding/removing of nodes).

- Scalability:
  The size of the datagrid can easily be modified by adapting the configuration on all the SRB nodes. A script-based automatic update mechanism is possible.

Disadvantages can be summarized to:

- Non-Uniform Concept:
  Each proxy installation has to be configured individually to fulfill the local requirements of the installation site. All software components of SRB have to support the proxy-usage.

- Complex Setup Procedure:
  Due to the individual configuration need of each proxy installation, a very complex installation and configuration task is needed for each instance of the application proxy.

An automatic proxy configuration by the use of authenticated sign-on is imaginable. Once a proxy is authenticated in the proxy-grid, all proxies in the grid can update their proxy-grid configurations. This feature can be used to maximize the scalability of the datagrid.

4.5.3 DMZ

In the *DeMilitarized Zone (DMZ)* szenario, the SRB nodes are placed within the DMZ of the hosting institution. A DMZ is the zone where all the servers reside that must be accessible from outside the institution. Generally, the DMZ is guarded by a first stage firewall. The second stage firewall is located behind the DMZ to protect the internal network from intruders.

The first stage firewall is weaker than the second stage firewall to ensure the accessibility for all computers located in the DMZ.

No access from outside should be granted behind the second stage firewall. Thus, the second stage firewall is a very strong instance, almost a one-way firewall. Usually, all the machines in the private network only have internal IP addresses that are not routed through the internet. The institutional router can do *Source Native Address Translation (SNAT)* to allow local machines to connect to the internet. After machine-internal routing, SNAT alters the source IP address to the IP address of the router, such that the contacted host knows his communication partner. If the contacted host sends answer packets, the router translates the destination IP address of the packet back to the original (internal) IP of the client and sends the packet to the corresponding machine. The SNAT takes place after the machine-internal routing, shortly before a packet leaves the machine.

The SRB node placement within DMZ ensures the accessibility from the outside world to this SRB node. An SRB server can be placed in the DMZ for real, virtual or two-stage. The subsequent paragraphs deal with these three different placement methods within the DMZ.

DMZ (real)

The *real* placement of an SRB node in the DMZ is shown in figure 4.7.

As shown in figure 4.7, the *real* SRB placement in the DMZ implies a physical presence of the SRB machine in the DMZ. All traffic to and from this machine only goes through the first-stage firewall with the same filters as all the other servers in the DMZ.

The advantages of this *real* placement are:

- Direct Access:
  A direct access to the SRB node machine is possible, such that the accessibility requirement for a firewall concept is fulfilled.

- Easy Realization:
  This concept can be easily realized. The SRB machine simply has to be placed in the *DMZ* and with a well configured firewall no additional configuration work is necessary.

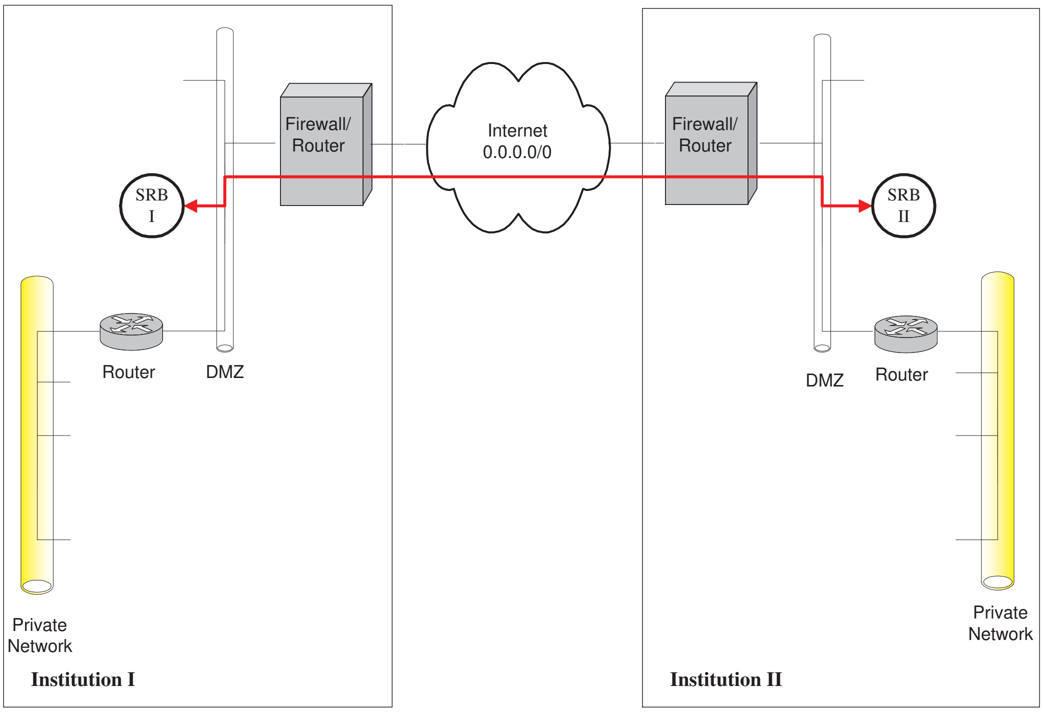

Figure 4.7: SRB Real DMZ Placement

- Uniform Concept:
  As the DMZ placement needs no additional administrative task, the concept can be used at every site without modifying the SRB node configuration.

The only drawback of the *real* placement in the DMZ is the lack of integration. Everyone is able to access SRB services and the SRB servers have to run isolated on a complete machine. As no access to the private net is granted, sharing of resources and dynamical enhancement of SRB storage with free and unused storage space in the private network space is impossible.

DMZ (virtual)

An SRB node can be placed as *virtual* in the DMZ of an institution. Figure 4.8 depicts this kind of placement.
To place an SRB node as *virtual* in the DMZ, the institution router has to be capable of *SNAT/DNAT* (explained below), while the SRB node is physically located behind second stage firewall in the private network of the institution. A direct connection similar to the *real* DMZ placement is emulated. The virtual placement is indicated by the blue line and the intercepted circle at every institution.

Source Native Address Translation (SNAT) alters the source-address of outgoing packets from the router. *Destination Native Adress Translation (DNAT)*

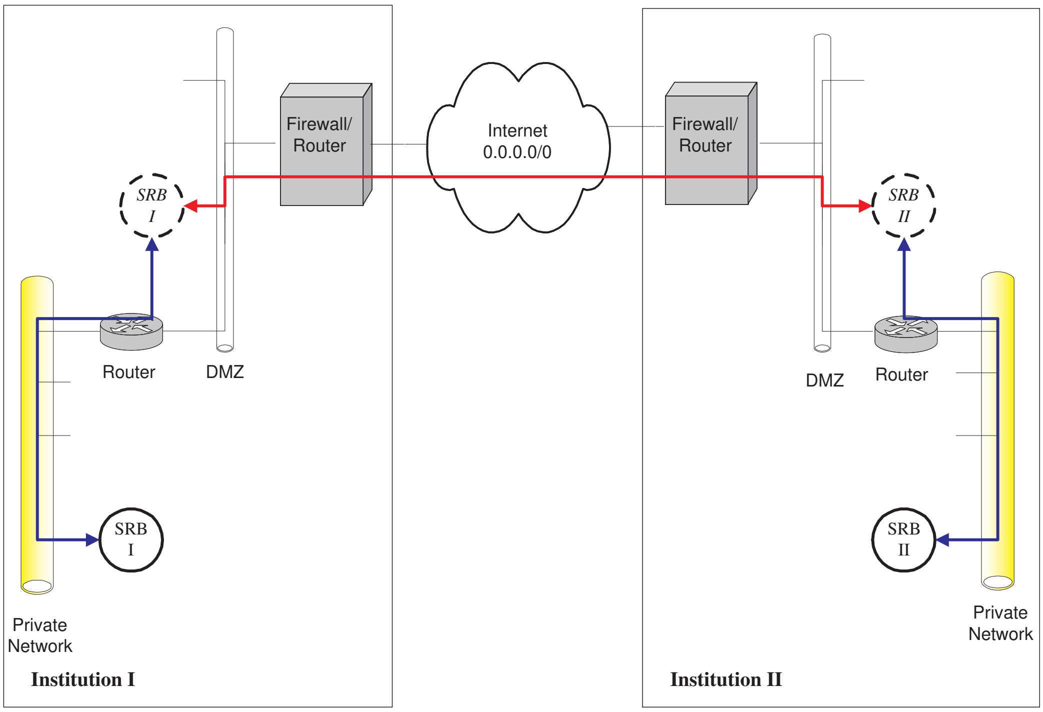

Figure 4.8: SRB Virtual DMZ Placement

alters the destination-address of incoming packets in the router.

First of all, the router has to get the external SRB node IP address. Whenever a packet for this IP address arrives at the router, the DNAT alters the destination IP address to the internal IP of the SRB node. This way, the SRB node receives all packets designated for that node. A special SNAT has to be done for the *virtual* placed machines. After machine-internal routing, the router does not change the response packet source address to his own address, as described above. It changes the source IP address of the packet to the external IP address of the SRB host. The advantages over the *real* SRB placement in the DMZ are:

- Direct access to the SRB storage server from both the virtual net and from the outside world is possible.

- Access to all private net resources for SRB is possible, which is useful for dynamically extending the storage space used by the SRB system.

Disadvantages compared to *real* placement can be seen as follows:

- Serious security problems can occur as direct access from outside world to the private net is granted.

- As a direct mapping from the private network to the external network takes place, this is a very special solution. A customized installation for every site is necessary.

- Under some circumstances, a connection slowdown can be noticed. An institution router is heavy loaded most of the time, so that any extra traffic will slow down the whole communication. This can be solved by port trunking [39] and upgrading the machines compute power.

This is the most flexible solution and very easy-to-use. Problems are due to the administration part of this concept. Not all grid groups in FhRG are responsible for their network management, but this solution depends on heavy administrative tasks. Hence, this solution is not practicable for each FhRG instance. However, this solution is compatible to the *real* SRB placement in the DMZ, so one can freely choose either of them.

DMZ (two-stage)

The *two-stage* placement of SRB in the DMZ is a combination of *real* and *virtual* SRB placement in the DMZ. Figure 4.9 shows the concept of *two-stage* placement.

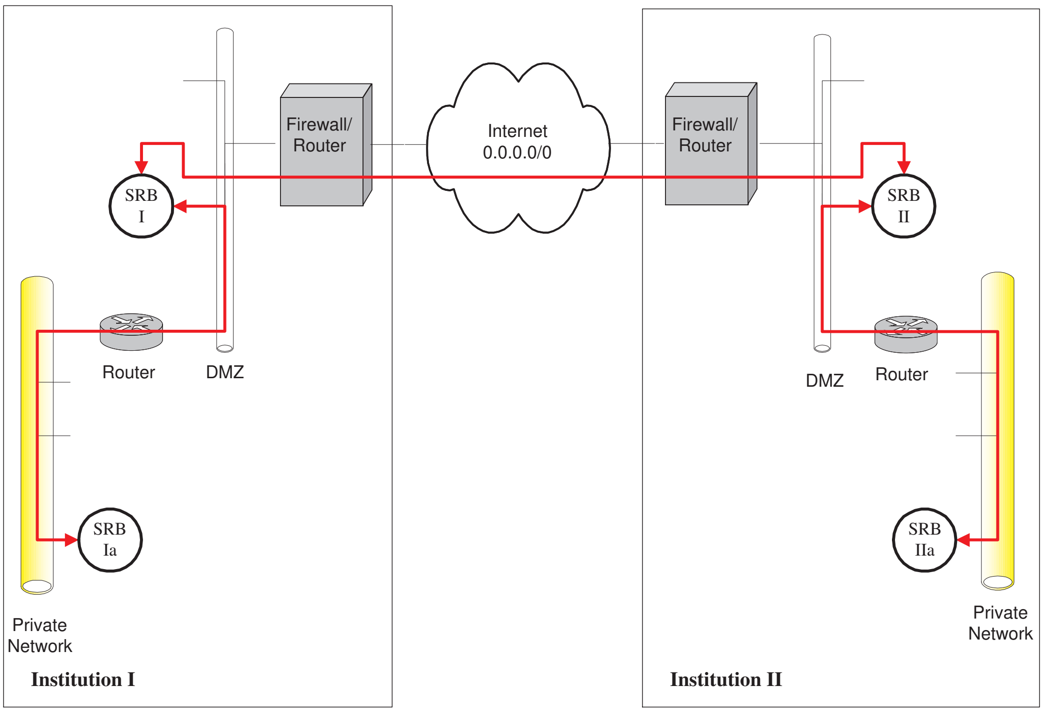

Figure 4.9: SRB Two-Stage Placement

Direct connections are shown by the red lines. The SRB components *SRB Ia* and *SRB IIa* are able to communicate with their respective *superior* components. The superior components *SRB I* and *SRB II* are able to communicate directly with their *inferior* components and all other superior components at other SRB sites. No direct connection between distributed (remote) inferior components or of a superior component with inferior components at remote sites is possible.

The SRB node *I* is located in the DMZ and can be directly accessed from the outside world. SRB node *Ia* can be accessed directly from the private net and from node *I*.
Two-Stage positioning brings some advantages and disadvantages. The advantages can be outlined as follows:

- Direct access to the datagrid from the both the private network and the internet.

- As no direct connection to the inferior nodes is being allowed, no security weakening of the actual firewall structure is necessary.

On the other hand, the disadvantages are:

- As there is no direct connection between all datagrid nodes, configuration problems may occur. Only limited data-transfers can be applied. Thus, with this concept, a datagrid hierarchy is established that is not wanted. If the administrator wants to cluster its SRB datagrid, the usage of zones is preferred. Due to these problems, this solution is not applicable to datagrids.

- No direct connection between all the *inferior* nodes and the MCAT server is possible, so the *two-stage* concept definitely can not be realized.

The *two-stage* scenario brings out some interesting features. With respect to the disadvantages, the scenario can not be realized in the required context.

If a multi-zone environment is developed, this solutions makes sense. Each institution has its own MCAT server, thus, the second point in disadvantages is solved.

4.5.4 Decision

Due to the disadvantages of the two-stage concept, this scenario can not be applied. So the decision is only between VPN, Proxy and DMZ (real/virtual) scenario.

As the SRB does not support proxy-based communication, this scenario can not be realized within the FhRG neither. With respect to the heavy administration task of the VPN solution, the DMZ placement of SRB is chosen.

In addition to the presented scenarios there are many other scenarios imaginable. The presented scenarios are only the most reasonable scenarios.

4.6 Connectivity Solutions

This section gives a short description of the APIs and the CLI that is offered by the SRB. The API/CLI intended usage areas are shown at the end of every API/CLI description.

The SRB API consists of the two main APIs (C-API and Java-API) and the fully featured CLI implementation (*S*-commands). To integrate the SRB within a client, the C or Java APIs can be used. The *S*-commands are intended for use within a shell.

SRB offers a fully functional C-API, that is capable of all the necessary file, user, resource, location, and metadata operations to operate the SRB effectively. With the help of this API, the SRB can be easily integrated within a client software written in C.

Java Api for Real Grids On Networks (JARGON), the pure Java API for SRB is not fully functional at this time. It fully supports file system operations and querying operations for accounts, resources, locations, and metadata. Setting operations for accounts, locations, or resources are not supported yet. Thus, programming of a client with a basic functionality (the file operations) is possible. As this API is under heavy development, new functionality will be added to *JARGON* soon.

The *S*-commands give a fully functional *Command Line Interface (CLI)*. They can only be used within a shell environment, thus an embedding within a C/Java client is not possible. These tools are intended to give an SRB user quick access to the SRB system, or to be used in shell-command high-level languages like PHP.

4.7 Use Cases

This section describes a few use cases of the most common scenarios encountered in the datagrid environment. The most users authenticate themselves in the datagrid and then work on their data.

Users can only authenticate with the help of a GSI-based authentication mechanism. To use this kind of authentication, each user needs a signed personal certificate and the certificate of the SRB servers certification authority together with the appropriate signing policy of the corresponding CA (subsection 4.1). If user authentication was successful, the system authorizes the user corresponding to the user entries in the *Access Control List* of the corresponding file-object. On the other hand, if the authentication fails, the user has no access to the datagrid system.

Once authorized, users are enabled to work on their data. The main operations that are done on the data are:

- Uploading files into the datagrid.

- Manipulating files in the datagrid.
- Downloading files from the datagrid.
- Generation of file replica.
- Removing files from the datagrid.

To bring files under the control of the datagrid system, the user is able to upload files from any file system into the datagrid volumes the user has write access to. After uploading, the files are under control of the datagrid management system.

Most operations that are done with files result in a file manipulation. If a user has write access to some data, the user is able to manipulate this data. Inconsistencies between the various replica of one file may appear. The datagrid system can be configured to automatically synchronize all existent replica of one file.

To get files from a datagrid, users can download them to any file system they have write access to. This step extracts data from the datagrid space back to the user space.

Users can generate replica of files in two ways. The replication command can be given explicitly to the datagrid management system. The other way to generate file replica is to define volumes that automatically replicate files on all physical resources of this volume. Synchronization of the replica can also either be done manually or automatically.

Files stored in the SRB datagrid can be opened by the user through an API call. The API call depends on the type of file object to open (file, LOBJ, etc.). After opening the file, various operations may be applied to the file (like reading, writing, seeking, status, etc.). Client programs that want to open files within a datagrid have to support the SRB API calls.

Files can be removed from a datagrid system, too. Files can be deleted by all users that have delete permission on the file.

Chapter 5

SRB Integration

This chapter describes both the installation/configuration procedure of SRB datagrid and the integration of the SRB datagrid in the FhRG environment.

An SRB datagrid consists of many SRB servers and at least one MCAT server. SRB uses metadata services of the MCAT server, thus the installation of this special MCAT server has to be done prior to the installation of the standard SRB server. The installation and configuration procedure of MCAT is described in section 5.1.

Section 5.2 describes the installation and configuration procedure of one prototypic SRB server.

5.1 Installation of the MCAT Server

This section describes the installation and configuration of one MCAT server. The installation procedure begins with a *naked* workstation, which has to be installed with a Linux operating system (see paragraph 5.1.2).

After successful system installation and configuration, the *DataBase Management System (DBMS)* server has to be installed. Paragraph 5.1.3 describes the installation of the *PostgreSQL* DBMS. Finally, as the system meets the minimum system requirements for the MCAT server, the MCAT server installation can be done (refer to paragraph 5.1.4).

5.1.1 Collecting Software

The SRB package was downloaded from the SDSC homepage:
http://www.npaci.edu/DICE/SRB/
To unpack the package, a special key is required. Thanks to Sheau-Yen Chen for giving me access to the required keys.

MCAT supports the Oracle, DB2, PostgreSQL and Sybase DBMS. As the PostgreSQL DBMS is used, the open source software PostgreSQL was down-

loaded from the PostgreSQL homepage:
http://www.postgresql.org

5.1.2 Linux Installation

The MCAT-enabled SRB server was installed on a new workstation. The hardware specification for this workstation is given in section A.1.

On the described system, the *Suse Linux Enterprise Server 8 (SLES8)* Linux distribution is installed. To improve the failure tolerance, the two hard disks are configured to run as a mirror set (RAID-1).

5.1.3 PostgreSQL Installation and Configuration

This paragraph describes the installation and configuration of the PostgreSQL DBMS. The configuration requirements are taken from the MCAT Installation Guide [40].

Installation
After unpacking the PostgreSQL archive, the software needs to be configured. Under consideration of the MCAT requirement for its underlying DBMS the PostgreSQL sources were configured and built. The configuration and building commands are explained in detail in section A.2.

Configuration
To configure the PostgreSQL DBMS the first command used is *initdb -D data*. This command generates all the necessary structures to run the PostgreSQL DBMS server. It generates a data directory *data/* where all the necessary structures are being initialized. The PostgreSQL runs completely in this directory and all databases are located in this data directory. PostgreSQL is capable of running multiple data directories, but this feature[1] is not used in this context.

Within the *data* directory there are two important configuration files called *postgresql.conf* and *pg_hba.conf*. *postgresql.conf* was modified to raise the number of maximum simultaneous connections and to enable the *syslog* logging-mechanism.

The file *pg_hba.conf* is the configuration file for the host-based authentication of PostgreSQL. As MCAT requires a TCP/IP connection to the local IP address (not the localhost IP address!) of the MCAT server it has to be subscribed in the *pg_hba.conf* file. The configuration of the file *pg_hba.conf* is discussed in detail in [41], chapter 8.

[1] for further information on multiple data directories, see [41] pages 336-339

After initializing the database structure and modifying all the configuration files, the PostgreSQL server can be started. This is done by the command:

postmaster -D data

Now, with the DBMS server running, a new database and user have to be created for the MCAT use.

As the MCAT server needs a PostgreSQL user named *mcat*, this user has to be created by the command *createuser*. This user needs full access to the database used by the MCAT server. The necessary database is created by the command *createdb -U mcat MCAT*, where MCAT is the name of the database and the *-U mcat* flag tells PostgreSQL to create the database as user *mcat*.

Now, all the structures needed by the MCAT server are created. The subsequent section describes the installation of the MCAT server.

5.1.4 MCAT Installation

With the PostgreSQL DBMS server running, the MCAT server is ready to be installed and configured.

After unpacking the SRB sources, they have to be configured and compiled. A detailed description of the configuration and compiling options is given in section A.3.

After successful compiling and installing of the MCAT server, the SRB working domain has to be created. This is done by several commands, for details see [26].

The creation of a working domain is followed by the creation of a new administrative user in this newly created domain. To do this, a predefined administrative account exists in the default domain.

MCAT is now completely installed and has to be configured for use in the FhRG. The next section 5.1.5 describes this configuration in detail.

5.1.5 MCAT Configuration

This paragraph shows some steps of the MCAT configuration procedure. First, the configuration tools of MCAT are presented. After the presentation of the administrative tools, some necessary parts of the configuration procedure are shown.

SRB offers several administrative tools to maintain the MCAT/SRB-based datagrid structure. The different tools are generated by the SRB compiling process. Each of these tool sets has a large set of functions that overlap partially, but some special tasks can only be done with the particular tool set that is designed with a special focus on the respective kind of operations. In other words, all of the tool sets are able to do a common set of operations, but none of the tool sets is able to accomplish all the necessary tasks to maintain an SRB datagrid.

The first administrative tool is the java-based *mcatAdmin* Tool. It gives the user the ability to manage SRB users and resources in an easy fashion. Figure 5.1 shows a screen shot of mcatAdmin with its user management window.

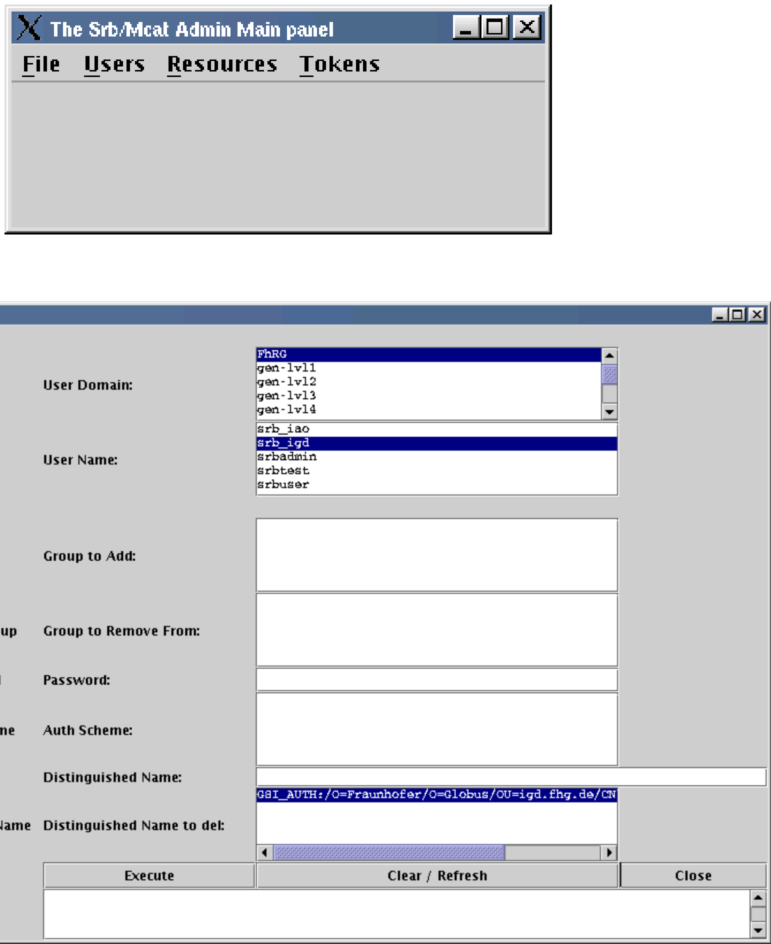

Figure 5.1: mcatAdmin

In addition to the mcatAdmin tool, SRB offers a set of shell commands, the *S* commands, for example *Sls, Smkdir, Srm, etc.* These commands are small client programs that utilize the *C*-API of SRB.

Finally, SRB offers some small administrative utilities that are generated at SRB compile time.

As first configuration step, a few users are created. This is done with the help of the mcatAdmin tool. One administrative user per SRB location is created, while all other accounts are normal user accounts. The only accepted authentication method is the *GSI* authentication (section 4.1). Therefore, a valid *Distinguished Name* has to be assigned to every user account.

All the SRB locations with the appropriate user accounts are created. An SRB location holds several SRB storage resources that are defined in the next configuration step. An SRB location is defined by a hostname and a parent location. The MCAT server is defined without a parent location (the *null* location). All other locations are defined as children of the MCAT location.

Once the locations are created, creation of SRB resources can occur. Many SRB resources are defined with the help of the mcatAdmin tool. An SRB resource is defined by a globally unique name, a valid SRB location where the resource will reside, a filesystem type (i.e. linux-unix filesystem, ADSM filesystem, PostgreSQL LOBJ, etc.), and a resource type (i.e. permanent, archival, cache, etc.). Further detail to resource types is given in paragraph 3.3.

Logical resources that aggregate one or more physical resources can be created with help of the mcatAdmin tool, too. In the first step, a logical resource is defined with all the physical resources that it aggregates. Additional resources can be added at runtime. Deleting of logical resources has to be done with the help of a command line tool.

McatAdmin enables an SRB administrator to define *compound resources*. *Compound resources* are resources that are not directly accessible, e.g. tapes. No compound resources are defined within the *FhRG* SRB installation.

Permission management on resources has to be done with the help of a command line tool name *modify_resource*. This tool is able to modify the permissions regarding to the respective resource. For example, after creating a physical resource, no user except members of the *sysadmin* group has write access to this resource. An administrator has to grant permissions for this resource to the user to allow using this resource.

5.2 Installation of the SRB Server

After successful installation and configuration of the MCAT-enabled SRB server, installation and configuration of standard SRB storage nodes can take place.

The SRB source distribution has to be configured and built. The detailed configuration and compiling options are shown in the section A.4.

As every SRB storage node has a different environment configuration, a global configuration can not be given. Only special parts of the configuration process can be shown globally. The special configuration options that depend on the nodes environment are shown in the next section.

An SRB server must set up some configuration files, like the *mcatHost* or the connection method. The most configuration parameters of an SRB server are given by the MCAT-enabled server at connection-time.

All the FhRG SRB servers are configured to contact the MCAT-enabled server at the *ITWM*. As the MCAT-enabled server at *ITWM* only accepts GSI-based authentication, the selected communication method is the GSI-based secure communication.

To establish a GSI-based authentication, each SRB instance (MCAT-enabled server, storage servers, datagrid users, etc.) needs an X.509 Certificate from a trustworthy *Certification Authority (CA)*. GSI-based authentication is described in detail at section 4.1. The necessary certificates were issued by the *FhRG Certification Authority*.

The storage on an SRB storage node is configured within the MCAT-enabled server. As the SRB server is an application that runs under non-root user credentials, the respective linux user that runs the SRB application must have write privileges to the designated SRB storage vaults.

5.3 Building a Datagrid

This section describes the procedure of spanning up a grid with a few datagrid nodes. The procedure of spanning up a grid of datagrid nodes consists of the node installing procedure and the configuration procedure of the different nodes to build one datagrid.

As the SRB storage nodes depend on the MCAT-enabled SRB server, installation of this MCAT-enabled server has to be done prior to installing the storage nodes. The MCAT-enabled SRB server is installed at the *ITWM* as the root node of the datagrid (see figure 5.2). To distinguish between the storage nodes, each node that is connected to the root node uses its own datagrid account in the FhRG domain. GSI-based authentication is required to connect to the root datagrid node. The GSI-based authentication mechanism is described in detail at section 4.1.

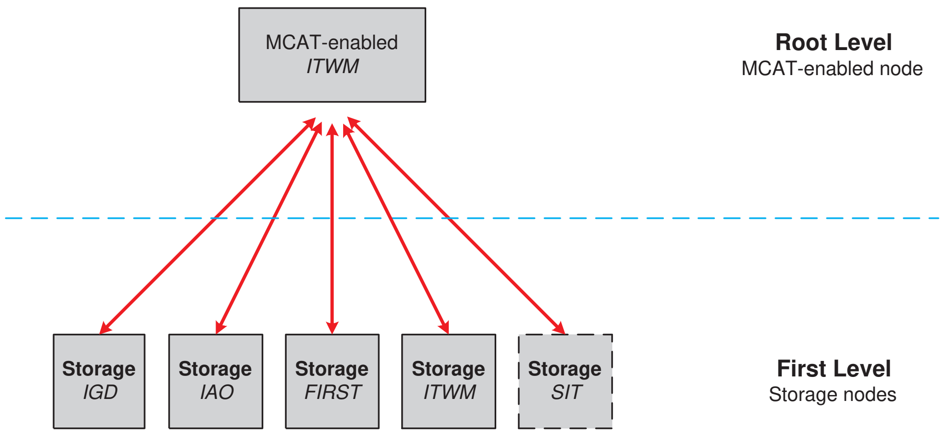

Figure 5.2: The Datagrid Tree

After successful installation and configuration of the MCAT-enabled SRB node, installation and configuration of the SRB storage nodes can take place. In the first installation stage, SRB is installed at the IGD, the IAO, the FIRST, and the ITWM institutions. All the SRB installations are set up as standard storage nodes.

A standard SRB storage installation within the FhRG contains at least one file system storage vault. As mentioned above, all standard installations of an SRB storage server are configured to have the MCAT-server at the ITWM as parent location. An SRB storage server hierarchy is created.

In the first installation stage, the SRB at SIT was skipped. A standard storage node installation at SIT is planned for the near future.

With the first installation stage completed, the datagrid consists of a two-level structure which allows the basic testing procedures necessary to analyze some of the SRB features in detail. The datagrid structure is shown in figure 5.2.

The second stage of installation addresses some special features of SRB as the support of LOBJs and tape-library devices.

To fulfill the necessary requirements for the LOBJ support, at least one storage node has to be configured with this support and a sufficient database must exist. As the IAO runs an Oracle-based database, the storage node at the IAO institute was reconfigured to support Oracle-based LOBJs. To enable LOBJ support, the SRB source code must be configured with the appropriate options. The installation at the IAO is configured with the *–enable-oralobj* option to enable support for LOBJs on an Oracle-based DBMS.

SRB supports tape-library devices. To use this feature, one storage node needs access to a tape-library device. The ITWM runs one ADSM-based [42] tape library device, thus this SRB feature is tested at the ITWM. Therefore, the standard SRB installation at the ITWM is extended by the use of the ADSM feature.

Chapter 6

Client Programming

This chapter describes the development of an SRB client program that uses the services of an SRB-based datagrid structure. The first section in this chapter gives a conceptual view to a general datagrid client application. In the second section, the client programming is shown. To finalize the chapter, the clients capabilities are analyzed and an outlook to future capabilities is given.

6.1 Client Concept

A datagrid client application should give the datagrid user an easy-to-use tool to administrate all the users data stored in the datagrid architecture. System administrators should be able to manage the datagrid system with the application completely. This includes user management, resource management, and location management tasks.

The problem to generate an easy client application are the different access permissions within a datagrid structure. An administrator has more access permissions than an ordinary datagrid user. Thus, an administrator is able to fulfill more tasks within the datagrid system. An application has to be generated that is able to generate different views respective to the user permissions.

To increase the usability of the client application, it should be a platform-independent software. Platform-independency implies that all users or administrators are able to connect to the SRB using this application, regardless of their respective operating system.

As users may handle with confidential data and to support a strong security concept that grants access only to permitted user, all communication between the client and all other datagrid nodes, including the web server where the client runs on, has to be encrypted.

Supported Platforms
As the client application should be platform-independent, a web-based approach

would be the best way to go. A web-based application runs in a web browser on the client side. The client application should be run without any restrictive software or hardware requirements.

As any modern operating system like *Linux* or *Micrososft Windows* supports the *Java Runtime Library* and SRB offers *JARGON*, a pure Java API, programming of a Java applet is chosen.

The Java applet can be easily integrated in a web page running on a web server that supports the use of *TLS* [12]. TLS-enabled web servers communicate with their clients through a strongly encrypted connection. Thus, a strong security is supported by the application.

To connect to the server, the client supports any web browser that supports the use of Java applets. All necessary classes needed to communicate with an SRB server are integrated within the client applet.

Applet Basics

A Java applet [43] is a graphical Java user program that can easily be integrated within a web page. The *HTML* [44] standard supports the use of applets within standard web pages.

Once a web page that contains an embedded Java applet is called, the applet code is transferred to the client. After transferring the necessary data, the Clients *Java Virtual Machine* executes the transferred code.

Unlike other programming languages, compiled Java source code is platform independent. The source code is compiled to a so called *byte code*, that is platform independent. At runtime, the Java Virtual Machine translates the byte code to machine code that is platform-dependent.

Client Structure

As mentioned above, the entry point for all users should be a uniform login mask to enter their user credentials. After successful authentication, the authorized user gets a customized view to the client that shows only the datasets that the user has access permissions to and the function that the user is allowed to apply to the datasets.

This method minimizes the displayed items to maximize the ease-of-use of the site for each user.

Client Appearance

The starting page of the web server is a page subdivided in two windows, the navigation bar window and a status window located at the bottom of the page. After clicking the *Login* button, a login mask is generated.

The parameters contained in the login mask are: (where the values in the brackets are the default values)

- SRB server host name (mcat.itwm.fhg.de)

- Port (5544)
- Domain Name (FhRG)
- User Name
- Password
- Authentication Scheme

The password is displayed as a row of stars.

If the login fails, an error message appears on the status window and the login page is displayed again. After a successful login, the main window and the toolbar are displayed.

The navigation bar controls the main window and the toolbar. If a category is selected in the navigation bar, the corresponding content is displayed in the main window. If user actions can be taken in the main window, a toolbar is displayed within the main window.

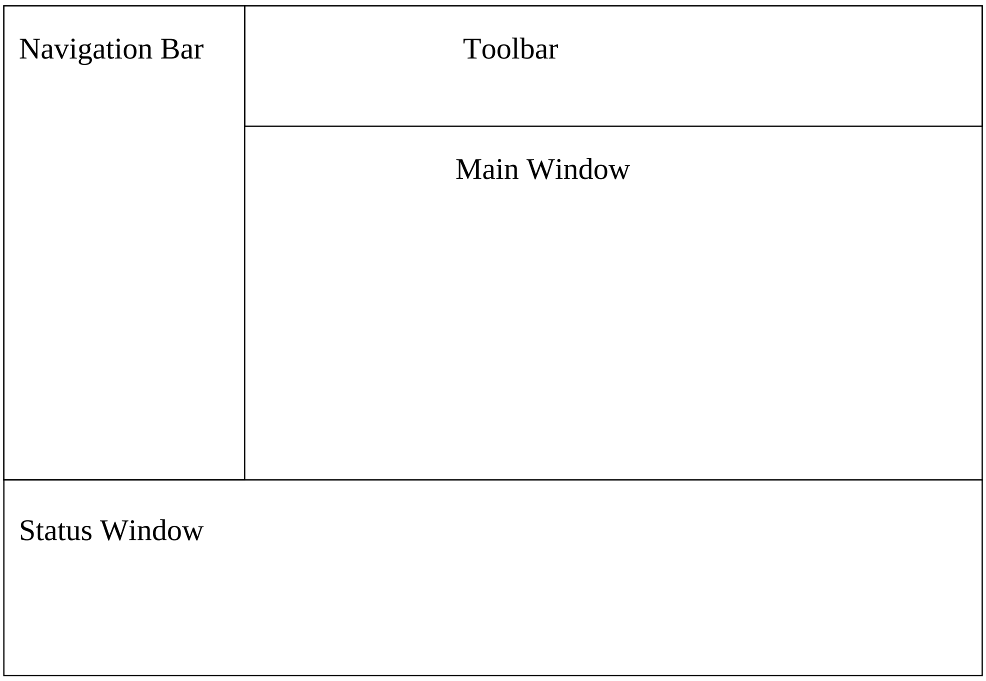

Figure 6.1: Schematic view of the client

To group the possible tasks, all tasks are subdivided into two main task groups, the user and the administrative tasks.

User tasks are all file operations (upload, read, write, replicate, download, etc.) file metadata operations (metadata creation, metadata reading, etc.), and account management (change password, change user information, etc.) regarding the own account. Administration tasks are all user and group management operations (add user, modify user, delete user, etc.), resource management operations (add resource, delete resource, security features, etc.), and location management operations (add location, delete location).

The navigation bar controls the system-level functions like login or logout. All user tasks are displayed within the toolbar of the main window.

As file operations work on files, all datasets accessible by the user are displayed in the main window.

In the case of an administrative login, the user administration link leads to a fully functional user administration task in the main window.

6.2 Programming

This section describes the steps of programming the datagrid client application. No code details are given, only main steps of the realization of the concept shown above are presented.

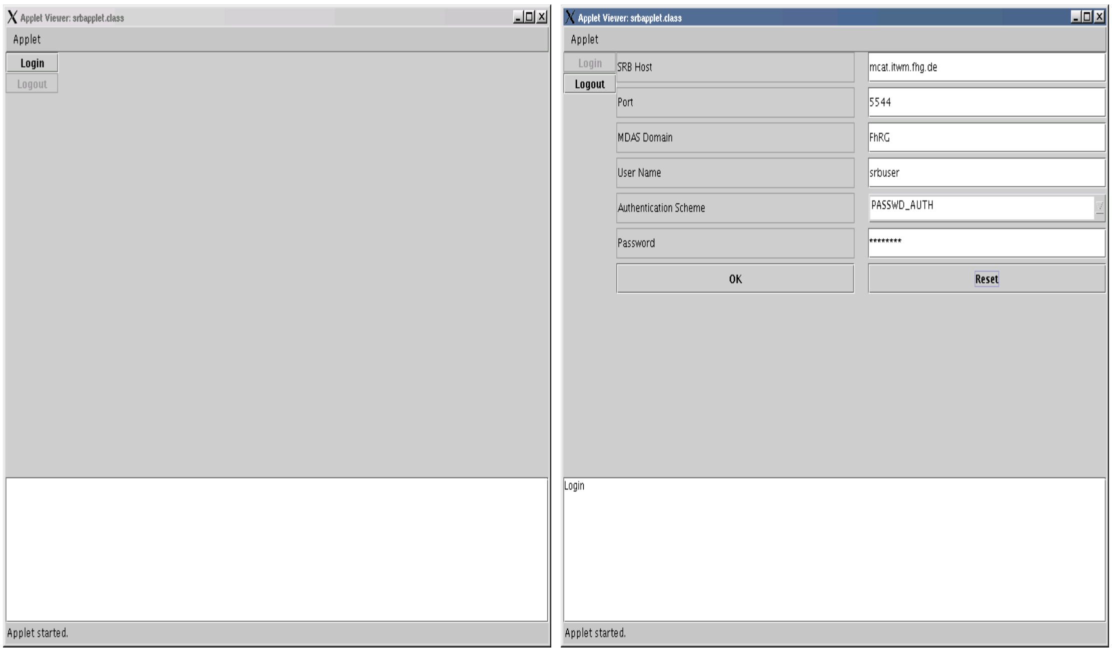

Figure 6.2: Starting & Login

Java supports the *Java Archiver (JAR)* which enables the programmer to package all files into one file. Aggregated files in a JAR archive are compressed, thus the traffic between server and client is quite low. Applets are not allowed to access the local file system on the client by default. This problem is solved by signing the applet with a private key using the *Public-Key Encryption* mechanism. By signing the applet, the client can choose whether to grant access to the local file system or not. Additionally, signing the applet secures the code. The principles of the *Public-Key Encryption* mechanism are described in detail in [30]. The realization of these principles in the Java programming language are described in [45].

To visualize the applet, the *Swing* [46] package is used. The Swing package is a graphical plugging package to Java. It enables the programmer to visualize complex graphics (file tree, buttons, etc.) in an easy way.

The programming is done with careful respect to the principles of object-oriented programming with strong use of modular and reusable objects. As the user tasks are a subset of the administrative tasks, some of the Java classes used for the administrative tasks are reused for programming the user tasks.

At startup time, the applet is being initialized. After pressing the *Login* Button, a login-mask is generated where some user credentials have to be entered. The starting and the login masks are shown in Figure 6.2.

After entering the user credentials and pressing the *OK* button, the SRB server is asked for user authorization. If the authorization is successful, the next screen is displayed showing all the tasks the user is able to do within the datagrid system.

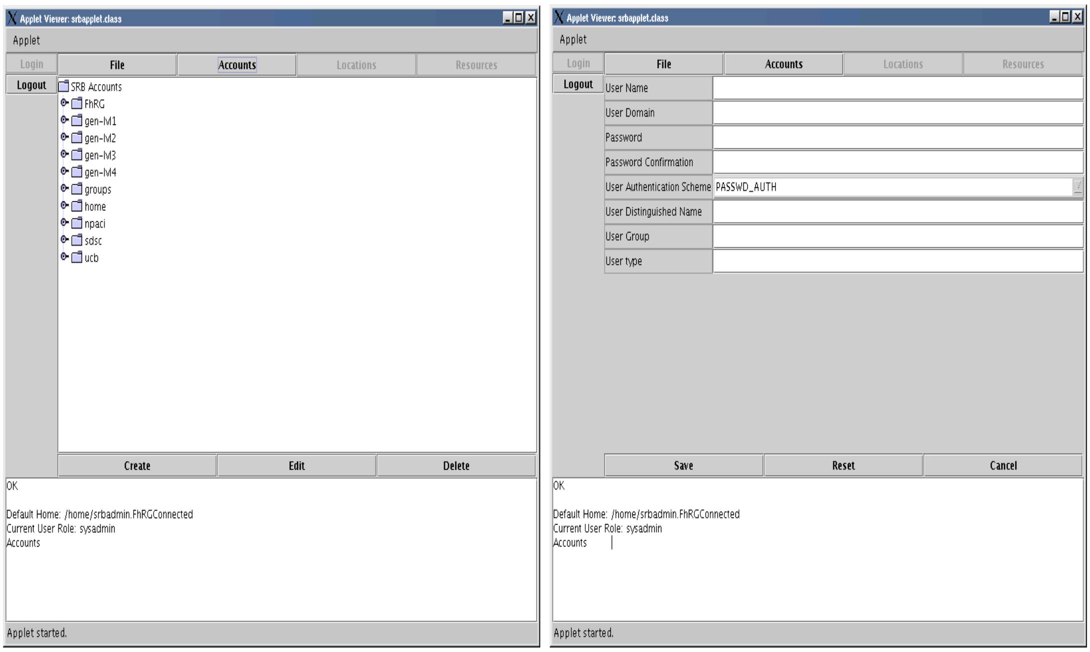

Figure 6.3: The administrator main window

Figure 6.3 shows the main window for the administrators. This window contains all buttons to access both the administrative and the user tasks. The left picture shows the user management window, the right picture shows the account creation dialog for an administrator.

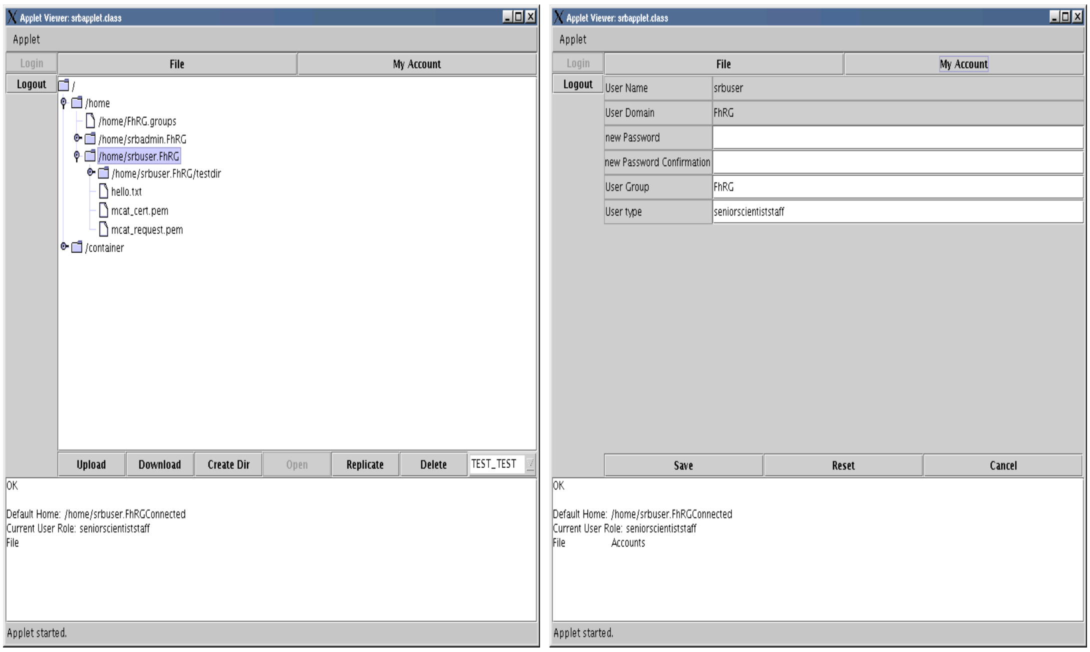

Figure 6.4: The user main window

Figure 6.4 shows the main window for a user login. The left screenshot shows the file management dialog, the right picture shows the user management dialog for the own SRB account.

6.3 Summary

In this section, the benefits and known limitations of the datagrid client application are shown. At the end, an outlook of future software features is given.

- The user-role dependant view of the main window works fine within the SRB client. After login, a user-role specific main window is generated. This window displays the tasks the user is able to apply to the SRB system.

- Due to technical limitations of the JARGON API, the administrative tasks are limited to file system and information querying tasks. No editing of users, resources, or locations is possible with the current version of the Jargon API.

- Nearly all file system operations are implemented, thus the user is able to proceed all allowed tasks on files. The *open file* task does not work yet, it is planned to be implemented in the next release.

- Due to technical limitations of the JARGON API, the account management task is not implemented yet. The user administration task will be implemented in the next release of the SRB client. At this time, only information querying about existing users is possible. Setting user data is not supported by the JARGON API, yet.

- No metadata operations are implemented, yet. This feature is to come in the next release of the client.
- Currently, only password-based authentication is supported. GSI-based authentication will be implemented in the next release of the SRB client.
- Uploading and downloading of multiple files with a single operation is possible, but the handling of complete directories is not implemented, yet. This will be implemented in the next release.
- The SRB client is not very robust against unexpected user input. This has to be implemented in future releases to improve the usability of the software.
- In future releases, the user will be enabled to define its own viewing parameters. These parameters will be stored as user defined metadata within the metadata catalog database.
- The client will be rewritten to achieve a more modularized structure of the software. With a fully modularized structure, implementation of a very flexible client structure will be made quite easy. Parts of the modularized software might be migrated to run at the server (e.g. as a servlet [47]) with a variable user-interface (for example applet, text-based, HTML-Code).

Chapter 7

Summary and Outlook

This chapter contains a short summary of the diploma thesis and an outlook of features and techniques to come.

7.1 Summary

This diploma thesis compares and evaluates several datagrid and *Distributed File System (DFS)* implementations. After evaluating the implementations, one of them is selected to be installed within the *Fraunhofer Resource Grid (FhRG)*.

After introducing the concepts of datagrids and DFS, both concepts are compared against each others. The comparison shows the different natures of the two concepts. Datagrids are intended to be used for worldwide inter-organizational data sharing within heterogeneous environments. The primary intention of the DFS concept is to share data within a single organization that offers a nearly homogeneous environment. Due to the nature of the FhRG as a heterogeneous environment, installing a datagrid software is prefered.

The comparison of the two concepts is followed by a comparative evaluation of the two datagrid implementations *European Datagrid (EDG)* and *Storage Resource Broker (SRB)*. To evaluate the two implementations, functional requirements are defined. With respect to the functional requirements, the evaluation shows no essential difference between the implementations EDG and SRB. Thus, non-functional aspects are used to select one software to be installed within the FhRG. The SRB project seems to have a clearer future than the EDG. At this time, the SRB project is not planned to be terminated, while the EDG project is terminated by the end of 2003. It will be continued within the *Enabling Grids for E-science in Europe (EGEE)* project funded by the European Union. In addition, the SRB is wide spread for a long time and expands very fast. Putting it all together shows a better prospect for the SRB project. Thus, an installation of the SRB datagrid software within the FhRG is prefered.

After selecting the SRB datagrid implementation, concepts for building a fast, reliable, scalable, and secure datagrid are developed. A global working concept is developed that defines the requirements for installing a datagrid within the FhRG. As the FhRG requires strong security, the grid security implementation *Grid Security Infrastructure (GSI)* is selected to be used within the datagrid software. To embed the datagrid software within the FhRG institutions, considerations about the possible firewall structure are made. To avoid conflicts with existing firewall structures and to minimize the administrative task, a placement of the datagrid software within the *DeMilitarized Zone (DMZ)* of each institution is chosen.

With a concept developed, installation of the SRB at four of five FhRG sites occurs. To enable datagrid users to access the datagrid in an easy way, a prototypic client is written. The client software is written as a Java applet to implement a platform-independent datagrid client software. This maximizes the usability of the client application with respect to the supported platforms.

7.2 Outlook

Many things are done within this diploma thesis, more things to come.

The SRB software is under heavy development, thus many improvements will come to this datagrid software. One example is the plan of the SRB developers to create an OGSA-compliant SRB with a stable OGSA specification release. With a focus on the OGSA-compliance, all offered services by SRB are going to be web services. The whole grid world (including the globus toolkit) is moving towards OGSA-compliant grid specifications. By this way, a standardized services architecture is created within the grid world.

Installation of the SRB is currently limited to a small set of distributed nodes. With the FhRG growing larger, the SRB-based datagrid will grow larger, too. At this time, the FhRG is in a testing phase. Once the tested concepts and software are proven to work fine in a grid environment, the FhRG will cover more institutions that share their resources with the grid. As a consequence, the shared datagrid space and the number of datagrid users will grow. Thus, the administrative task will get the more and more complex. The good scalability of the datagrid management keeps the administrative effort controllable.

Currently, the EDG and the SRB a two separate products that can not cooperate at this time. One plan of the FhRG project is to offer a modularized datagrid system that supports cooperation with other datagrid architectures of partner projects. As the Fraunhofer ITWM participates in the EGEE project, a connection between EDG and SRB is aspired. If an EDG-based datagrid like the EGEE wants to cooperate with the FhRG, the connection between both datagrid architectures will be possible. SRB and EDG are about to become OGSA-compliant grid services. With the standardized services architecture required by the OGSA description, connecting the two datagrid architectures can easily be implemented.

This plan is to be seen as a long-term project.

The client developed in chapter 6 will be expanded with many new features, as described in the summary of chapter 6. In the near future, the missing functionality will be implemented. When the main implementation work is finished, the client will be tested by the FhRG grid community. One goal of the client is to deliver a complete datagrid management tool which aggregates all necessary functions to manage a datagrid with one tool.

Appendix A

Hardware and Compile Parameters

A.1 MCAT Hardware Specification

To install the MCAT server, a blank workstation manufactured by transtec[1] is used. The system specifications for the 'transtec 1001 Xeon' are:

- CPU:
  1 * Intel Pentium 4 Xeon 2.6 Ghz
- Mainboard:
  Supermicro P4DPR+ Dual Xeon
- Memory:
  2GB ECC DDR-SDRAM
- Harddisks:
  2 * Seagate Cheetah Ultra 320 SCSI ST336607LCA 36.7GB
- Network:
  1* Intel 82550 Fast Ethernet Controller
  1* Intel 82554GC Gigabit Ethernet Controller

A.2 PostgreSQL Installation Options

The PostgreSQL DBMS was compiled and installed with the following options:

- ./configure –prefix=/var/lib/pgsql –with-openssl –with-maxbackends=256
  –with-odbc –with-pam

[1] transtec: www.transtec.de

- make
- make install

configure generates the *makefile* needed by the make command. This makefile contains all the necessary options to compile the source files. As MCAT requires ODBC support, the ODBC has to be compiled in (–with-odbc option).

The *–prefix=* tells the *make* command to install the software in an alternate directory. This is necessary for security reasons. If you install and run a program as a non-privileged user, the program has only the user permissions of the ordinary user and can not manipulate the operating system. As non-privileged users do not have write access to the standard system directories, the installation must be done in a directory that is writable by the user installing the software.

To improve the connectivity of the MCAT server, the maximum number of simultaneous connections was set to 256 (*–with-maxbackends=256* option).

Setting the flag *–with-openssl* enables support for the open-source TLS implementation *OpenSSL* [48]. The option *–with-pam* enables *Pluggable Authentication Modules (PAM)* [49] support. These two features increase the security level of the PostgreSQL DBMS.

Doing the command *make* compiles the source code. The *make install* command installs the just compiled binaries to the target directory given with the *–prefix* option to *configure*.

A.3 MCAT Installation Options

The MCAT-enabled SRB server was configured and compiled with the following commands:

- ./configure –enable-psgmcat –enable-psglobj –enable-accsctrl
  –enable-javagui –enable-gsi-auth
  –enable-psghome=/var/lib/pgsql
  –enable-installdir=/var/lib/pgsql/srb
  –enable-jdkhome=/usr/lib/java
  –enable-globus-location=/var/lib/pgsql/globus
- make
- make install

To generate the Makefile needed by the *make* command, the *configure* command needs some important flags set. These flags are described below:

- *enable-psgmcat* enables the MCAT functionality of the SRB server. It sets PostgreSQL as the underlying DBMS server, too.

- *enable-psglobj* enables the *large-objects* support for the PostgreSQL DBMS.
- *enable-accsctrl* enables access control of the metadata.
- *enable-javagui* instructs the *make* command to compile the java-based administration tools for SRB/MCAT.
- *enable-gsi-auth* enables support for the GSI authentication mechanism discussed in section 4.1. This option requires an existing GSI installation.

The subsequent flags inform the compiler of the locations of the selected component, where /var/lib/pgsql is the home directory of the user *pgsql*. The *configure* command supports much more options to be set, the described ones are the important flags. All other flags are less important.

A.4 SRB Installation Options

All the *standard* SRB servers are configured and compiled with the following commands:

- ./configure –enable-gsi-auth –enable-javagui
  –enable-installdir=/home/pfs4/srb
  –enable-jdkhome=/usr/lib/java
  –enable-globus-location=/home/pfs4/srb
- make
- make install

To generate the Makefile needed by the *make* command, the *configure* command has some important flags set. These flags are described below:

- *enable-javagui* tells the *make* command to compile the java-based administration tools for SRB/MCAT.
- *enable-gsi-auth* enables support for the GSI authentication mechanism discussed in section 4.1. This option requires an existing GSI installation.

The subsequent flags inform the compiler of the locations of the selected component, where /home/pfs4/srb is the home directory of the user *srb*.

If the SRB installation requires a special range of ports to be used by SRB, the options *–enable-commports*, *–enable-commstart=<startport>*, and *–enable-commnum=<number of ports>* allow customizing the port ranges used by SRB for data transfers.

List of Figures

Bibliography

[1] European Union.
The DataGrid Project.
http://eu-datagrid.web.cern.ch/eu-datagrid/.

[2] Avaki Corporation.
Avaki : Home.
http://www.avaki.com.

[3] San Diego Supercomputer Center, University of California.
SDSC Storage Resource Broker.
http://www.npaci.edu/DICE/SRB/.

[4] *Particle Physics Data Grid project (PPDG).*
http://www.ppdg.net.

[5] *GriPhyN - Grid Physics Network.*
http://www.griphyn.org/.

[6] *Computational Grids : The Clipper Project.*
http://www-itg.lbl.gov/Clipper/.

[7] GGF.
Global Grid Forum.
http://www.ggf.org.

[8] I. Foster, D. Gannon.
The Open Grid Services Architecture Platform.
Global Grid Forum, 2003.
http://www.gridforum.org/Meetings/ggf7/drafts/draft-ggf-ogsa-platform-2.pdf.

[9] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, P. Vanderbilt.
The Open Grid Services Architecture InfrastructureVersion 1.0.
Global Grid Forum, 2003.
http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf.

[10] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana.
Web Services Description Language (WSDL) 1.1.
Ariba, International Business Machines Corporation , Microsoft, 2001.
http://www.w3.org/TR/wsdl.html.

[11] Housley et. al.
RFC 3280: Internet X.509 Public Key Infrastructure, Certificate and Certificate Revocation List (CRL) Profile.
The Internet Society, 2002.
http://www.ietf.org/rfc/rfc3280.txt.

[12] Dierks,Allen.
RFC 2246: The TLS Protocol, Version 1.0.
Certicom, 1999.
http://www.ietf.org/rfc/rfc2246.txt.

[13] Fraunhofer Gesellschaft.
Fraunhofer Resource Grid.
http://www.fhrg.fhg.de.

[14] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler.
Extensible Markup Language (XML) 1.0 (Second Edition).
W3C, 2000.
http://www.w3.org/TR/REC-xml.

[15] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Tuecke.
RFC 2246: The TLS Protocol, Version 1.0.
ANL, 2001.
http://www-fp.mcs.anl.gov/dsl/GridFTP-Protocol-RFC-Draft.pdf.

[16] *OpenAFS.*
http://www.openafs.org.

[17] Integration Team (WP6).
DataGrid - EDG Users' Guide.
http://marianne.in2p3.fr/datagrid/documentation/EDG-Users-Guide.pdf.

[18] Integration Team (WP6).
DataGrid - Installation Guide.
http://marianne.in2p3.fr/datagrid/documentation/EDG-Installation-Guide.pdf.

[19] J. Casey, P. Kunszt, E. Laure, H. Stockinger, K. Stockinger.
DataGrid - User Guide for EDG Replica Manager 1.4.8.
http://proj-grid-data-build.web.cern.ch/proj-grid-data-build/edg-replica-manager/user-guide/edg-replica-manager-userguide.pdf.

[20] W. H. Bell, G. McCance.
DataGrid - Spitfire User Guide v2.1.
http://edg-wp2.web.cern.ch/edg-wp2/spitfire/doc/v2.1/SpitfireUserGuide_v2.1.ps.

[21] A. Rajasekar, M. Wan, R. Moore, G. Kremenek, T. Guptil.
Data Grids, Collections, and Grid Bricks.
San Diego Supercomputer Center, University of California, San Diego, CA 92093-0505.

[22] A. Rajasekar, M. Wan, R. Moore.
MySRB & SRB.
San Diego Supercomputer Center, University of California, San Diego, CA 92093-0505.

[23] M. Wan, A. Rajasekar, R. Moore, P. Andrews.
Simple Mass Storage System for the SRB Data Grid.
San Diego Supercomputer Center, University of California, San Diego, CA 92093-0505.

[24] San Diego Supercomputer Center, University of California, San Diego, CA 92093-0505.
SRB - The Storage Resource Broker (Version 2.0.0).
http://www.npaci.edu/DICE/SRB/CurrentSRB/SRB.htm.

[25] A. Rajasekar.
MCAT - A Meta Information Catalog (Version 1.1).
San Diego Supercomputer Center, University of California, San Diego, CA 92093-0505.

[26] San Diego Supercomputer Center, University of California, San Diego, CA 92093-0505.
SRB Projects.
http://www.npaci.edu/dice/srb/Projects/main.html.

[27] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, D. Winer.
Simple Object Access Protocol (SOAP) 1.1.
DevelopMentor, International Business Machines Corporation, Lotus Development Corporation, Microsoft, UserLand Software Inc., 2000.
http://www.w3.org/TR/SOAP/.

[28] M. Wahl, T. Howe, S. Kille.
RFC 2251: Lightweight Directory Access Protocol (v3), 1997.
http://www.ietf.org/rfc/rfc2251.txt.

[29] *Globus: Grid Security Infrastructure.*
http://www.globus.org/security/overview.html.

[30] A. Menezes, P. van Oorschot, S. Vanstone.
Handbook of Applied Cryptography.
CRC Press, 1996.

[31] J.Postel, J.Reynolds.
RFC 959: File Transfer Protocol (FTP).
ISI, 1985.
http://www.ietf.org/rfc/rfc959.txt.

[32] M. Horowitz, S. Lunt.
RFC 2228: FTP Security Extensions.
Cygnus Solutions, Bellcore, 1997.
http://www.ietf.org/rfc/rfc2228.txt.

[33] P. Hethmon, R. Elz.
RFC 2389: Feature negotiation mechanism for the File Transfer protocol.
Hethmon Brothers, University of Melbourne, 1998.
http://www.ietf.org/rfc/rfc2389.txt.

[34] R. Elz, P. Hethmon.
Extensions to FTP.
Prince of Songkla University, Hethmon Brothers.
http://www.ietf.org/internet-drafts/draft-ietf-ftpext-mlst-16.txt.

[35] Massachusetts Institute of Technology.
Kerberos: The Network Authentication Protocol.
http://web.mit.edu/kerberos/www/.

[36] H. Stockinger, S. Muzaffar, F. Donno, A. Konstantinov, A. Samar, E. Laure.
Grid Data Mirroring Package - User Guide for GDMP 4.0.
http://cmsdoc.cern.ch/cms/grid/userguide/userguide-gdmp-4.0.ps.

[37] Linux Networx.
Linux Networx.
http://www.lnxi.com.

[38] B. Gleeson, J. Heinanen, A. Lin, G. Armitage, A. G. Malis.
RFC 2764: A Framework for IP Based Virtual Private Networks.
The Internet Society, 2000.
http://www.ietf.org/rfc/rfc2764.txt.

[39] *IEEE 802.3ad Link Aggregation Task Force.*
http://grouper.ieee.org/groups/802/3/ad/.

[40] San Diego Supercomputer Center, University of California, San Diego, CA 92093-0505.
README.MCAT.INSTALL.
http://www.npaci.edu/dice/srb/install/README.MCAT.INSTALL.

[41] J. C. Worsley, J. D.Drake.
Practical PostgreSQL.
O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 1st edition, January 2002.

[42] *adsm.org :: TSM/ADSM Resources.*
http://www.adsm.org.

[43] *Applets.*
http://java.sun.com/applets/.

[44] *W3C HTML Home Page.*
http://www.w3.org/MarkUp/.

[45] B. Eckel.
Thinking in Java.
Mindview Inc., 3rd edition, 2003.

[46] M. Loy, R. Eckstein, D. Wood, J. Elliott, B. Cole.
Java Swing.
O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2nd edition, November 2002.

[47] J. Hunter, W. Crawford.
Java Servlet Programming.
O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2nd edition, April 2001.

[48] *OpenSSL.*
http://www.openssl.org.

[49] *A Linux-PAM page.*
http://www.kernel.org/pub/linux/libs/pam/.