Data Management System for grid and portal services

Piotr Grzybowski[1], Cezary Mazurek[1], Paweł Spychała[1], Marcin Wolski[1]

[1]Poznan Supercomputing and Networking Center, ul. Noskowskiego 10,
61-704 Poznan, Poland
{piotrgrb, mazurek, spychala, maw}@man.poznan.pl

Abstract: Poznan Supercomputing and Networking Center has been involved in continuous research concerning the development of grid-portal environments. In the scope of the PROGRESS project, we have designed and implemented the Data Management System (DMS). The DMS is an example of a service which might support other grid and portal services. Its main task is to store and manage computational data. One of the additional functions is proxy to other databases and data accessed from the Internet. The next function is mirroring the external database. In this paper we present the DMS architecture and the functionality of its main modules. The DMS provides an interface for other clients. The client (e.g. external service) can be a broker or special DMS client e.g. computational web portal. The DMS can also manage several replicas of data and allow access and mirror external databases. Data can be stored in several ways: in computer filesystems, in databases or on tape storage systems. The main functionality of the presented system has already been implemented and it was deployed as a part of the testbed presentation on SC2002 exhibition.

1. Introduction

Data management is one of the core services in the Grid systems. Grid computing is an important new field focused on large-scale resource sharing. We integrate both those issues within the PROGRESS project aimed at creating an "Access Environment to Computational Services Performed by Cluster of SUNs" [6]. The PROGRESS is currently under development in Poznan Supercomputing and Networking Center (PSNC). The initiative, undertaken in the scope of the PIONIER National Program [13], started in 2001 and is funded by the State Committee for Scientific Research and Sun Microsystems Poland. The demo version of the PROGRESS HPC Portal testbed, including DMS, was presented at the Supercomputing 2002 (SC2002) conference in Baltimore. This project is developed by PSNC in co-operation with: University of Mining & Metallurgy in Kraków and Technical University of Łódź. In the following chapters we describe the known Data Management

Environments (Ch. 2) and the overall PROGRESS architecture (Ch. 3). Further on we present the DMS architecture (Ch. 4) and discuss our approach to transferring data files in the DMS (Ch. 5). Future work is described in chapter 6.

2. Data Management Environments

One of the most advanced concepts for data management is being developed within the Gryphin project. The Gryphin (Grid Physics Network) [1] project is team collaboration that will implement the first Petabyte-scale computational environments for data intensive science. GriPhyN deploy computational environments Petascale Virtual Data Grids (PVDGs) that meet the data-intensive computational needs. Data analysis for huge computing tasks causes that computing and storage resources required should be distributed for both technical and strategic reasons across national centers, regional centers, university computing centers, and individual desktops. The GriPhyN collaboration proposes to carry out the necessary computer science and validate the concepts through a series of staged deployments, ultimately resulting in a set of production Data Grids. GriPhyN pursues a program of fundamental IT research focused on realizing the concept of Virtual Data. Virtual Data encompasses the definition and delivery to a large community of a (potentially unlimited) virtual space of data products derived from experimental data. In this virtual data space, requests can be satisfied via direct access and/or computation, with local and global resource management, policy, and security constraints determining the strategy used.

One of the most interesting systems dedicated to managing data resources was created by the National Partnership for Advanced Computational Infrastructure (NPACI). The SDSC Storage Resource Broker developed there (SRB) [2] is a client-server middleware that provides a uniform interface to connect to heterogeneous data resources over a network and access replicated data sets. In conjunction with the Metadata Catalog (MCAT) it allows to access data sets and resources based on their attributes rather than their names or physical locations. SRB is presented as a package that is a sort of a distributed file system with some elements of data grid management. Using SRB requires using supplied C-style API. The system has been developed for several years and used in a number of projects, including Gryphin (Grid Physics Network [11]).

The Storage Resource Management (SRM) [10] Middleware Project (related to the Gryphin) develops common application interfaces to data and deploys them in real grid applications. The project targets both tape-based and disk-based systems and designs their interfaces so that they inter-operate. The purpose of this project is to address the problem of managing access to large amounts of data distributed over the sites of the network. Architecture of Storage Resource Managers (SRMs) assumes that SRMs are associated with

each storage resource on the grid in order to achieve coordinated and optimized usage of these resources. The term “storage resource” refers to any storage system that can be shared by multiple clients. The goal is to use shared resources on the grid to minimize data staging from tape systems, as well as to minimize the amount of data transferred over the network. The main features of this system support local policy (for each storage resource), temporary locking files, advance reservation, dynamic space management and estimates for planning.

The fundamental objective of another data-bound project - the DataTAG project [9] - is to create a large-scale intercontinental Grid testbed involving the European DataGrid project, several national projects in Europe, and the related Grid projects in the USA. The main goal of this project is to work out transparent access to the massively distributed computing infrastructure that is needed to meet the challenges of modern data intensive applications.

Another DME that can be quoted as an example of such system is Spitfire [3]. Spitfire is a project of Workpackage 2 [4] within the European Data Grid Project [5]. It provides a Grid enabled middleware service to access relational databases, using the Spitfire Client- Server library modules and command line executables. Access to various RDBMS systems is possible through standard protocols and describing their interfaces. It defines appropriate SOAP-based services which are realizing internal functionality. User gets client-side APIs provided for java and c++ for the SOAP-enabled interfaces.

All of those presented systems are in some ways similar to the Data Management System, which is being developed in PSNC. When creating a specification of DMS [6] for the PROGRESS project, the working team made a few assumptions. Decisions that were taken led to the development of a project of DMS that should be able to work with different kinds of data storage resources (similarly to other described systems). Storing data will be realized basing on logical structures similar to a standard file system structure (with catalogs and flat-files enhanced with the links and containers functionality). The logical structure is independent of the physical storage resource type. Data stored within the PROGRESS DMS can be described by additional information collected into the meta-data repository. Flexibility of assessing the data requires that they are available using different kinds of standard data transfer protocol (e.g. FTP, GridFTP [8] and GASS [7]) and should not demand to use a special sort of client libraries to access the stored data. This has led to working out a common interface that allows to get data from (and put them to) DMS without additional necessity of changing the code of client applications. The other important problem to solve during projecting and implementing DMS was flexibility of using different kind of resource storing solutions. Having in mind this assumption the PROGRESS work team have decided to use modular internal architecture of DMS with various modules realizing operations on various kinds of storage systems (e.q. standard flat-file systems, database systems or archiving systems like tape

archive system). All those data containers work basing on common interface; thus they are services by the DMS in the same way as well. Using the SOAP standard and java coding within internal infrastructure makes this DMS useful and easy to modify for a specific solution. The functionality of DMS meets the requirements of grid environments like all other DME's do; however, its architecture and implementation through their flexibility assure adopting any of open architectures for grid services (e.g. OGSA [15]).

3. DMS as a part of the PROGRESS project

The PROGRESS project architecture provides the implementation of five components which together form an example of grid-portal architecture. One of the crucial items of the PROGRESS grid-portal achievements is Grid Service Provider (GSP). The main task of the GSP is providing a flexible and comfortable way of accessing the grid resources. The GSP serves as a mediator between the user interfaces, such as the WP, and the grid. Along with the GSP, four other modules constitute the system: the web portal (WP), the migrating desktop (MD), the data management system (DMS) and the grid resource broker (GRB). The PROGRESS grid is powered by three Sun Fire 6800 servers, placed in Poznan and Cracow and connected by the Polish National Research Network.

The GRB enables the execution of PROGRESS grid jobs in a real distributed cluster of three Sun computers. The cluster is managed by Sun Grid Engine software with Globus deployed upon it. The GRB provides two interfaces: a CORBA interface and a web services one. The SC2002 PROGRESS demo used the former one, but this will be changed to the latter in the final deployment. Grid job definitions are passed to the GRB in form of an XRSL document and the GRB informs the GSP about events connected with the execution of a job (start, failure or success).

The PROGRESS GRB module uses the DMS to store the input and output files of computed jobs. The DMS includes a data broker which handles all requests. Interface of this module is based on the 'web services' technology. The DMS can be equipped with three data containers – file system, database system and tape storage system. The DMS prepared for SC2002 demo was using only the first one; work on the rest of them is in progress. A data file is referenced within the DMS with a universal object identifier. The identifier allows obtaining information on the location of the file. Each file may be downloaded or uploaded using one of three possible protocols: FTP, GASS [7] or GridFTP [8].

The PROGRESS architecture and the communication manners between modules are illustrated on figure 3.1.

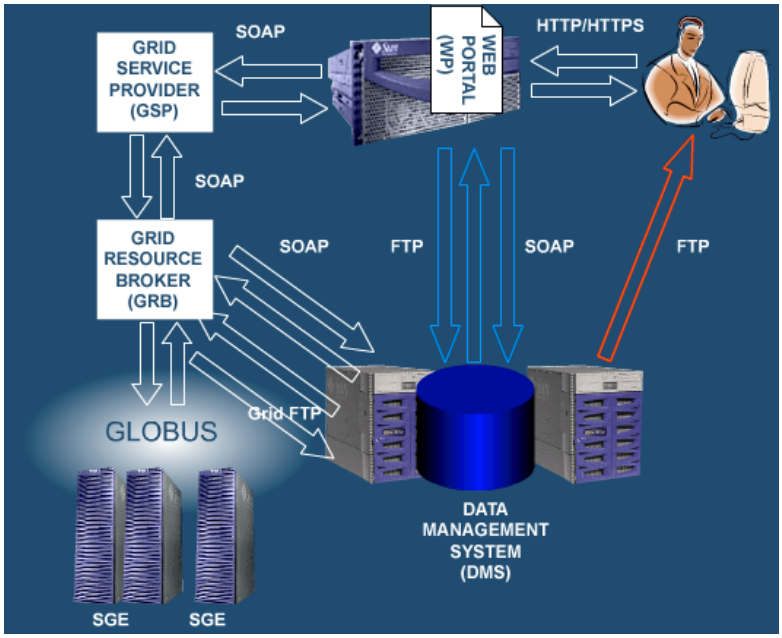

Fig. 3.1 The PROGRESS architecture and communication

The DMS data files are organized as a directory tree, therefore a natural user interface for management of the DMS files is a directory browser. Such a browser is a part of the PROGRESS WP. It allows adding, renaming and deleting directory and file entries. Addition of files gets a form of a web browser file upload. After a file is uploaded to the web server where the WP is installed, the WP service responsible for contacts with the DMS adds a file to the metarepository (using a proper data broker method) and stores the file using the FTP protocol (more detailed in chapter 5). The files may also be retrieved with the usage of the WP (Web Portal). The WP service, knowing the file reference, starts the retrieve sequence by learning the current FTP URL for the file. The URL is then passed to the user in a form of a web page link. The user completes retrieving the file by downloading data from the indicated link.

The PROGRESS grid job descriptor contains references to job input and output files as well as to binary executable of the application to run. While the input files are simply chosen from the directory tree by the user and their references are stored in the descriptor of the job being built, the output files must be created first. So, the user forces the WP to add a file entry in the desired directory and save the new file reference in the job descriptor. The reference to the application descriptor is entered through a user choice from a drop-down list on a web page; the user decides what application to run. After the job is submitted to the GSP, its Job Submission service prepares the XRSL description of the job and passes it to the GRB for execution.

After successful parsing of the description the GRB retrieves the required input files and the application binary code. It contacts the DMS data broker and requests the GridFTP URLs. Then the data are downloaded. After the retrieval of files the job is executed in the grid. The completion of the job indicates that the results should be now stored in the DMS. The GRB performs the store sequence for all the job output files. From now on, the completed computing experiment results are available to user.

Also, a few features that make the DMS system powerful and universal are worth mentioning. The system universality is revealed in its possibility to integrate different kinds of storage resources, enable access to data using different transport protocols, and enhance the functionality of the DMS by preparing new modules compatible with common interface. Other features include the possibility to build metadata schema and create matching access policy fulfilling the applied assumptions. The need of data integration in grid environment, where data are stored under different locations and on different storage systems, resulted in the creation of data sets that can be understood as a virtual data catalog.

The Metadata module of the DMS allows adding some extra information assigned to the real data stored using a traditional storing system. This feature creates a possibility of integrating data resources of different kind and for different purpose in a common schema. On the other hand, the subsets of data that share common features can be assigned to unique category. It is done by selecting or even creating a new metaschema.

A separated module devoted for implementing direct interface to the storage system – called the Data Container – provides uniform access to the storage resources independently of the internal structure of this resource. Prepared modules can hold data using flatfiles on standard file systems or data files stored in tape archive systems, or keep them in the database records. In the beginning we prepared a Data Container module that works on flatfiles and supports the FTP, GridFTP [8] and GASS [7] – both secure and standard versions.

The modularity of the internal structure of the DMS allows creating an open system. The data sets can be stored on different media types (as mentioned above) alternatively data can by acquired from other system such as SRS. (Sequence Retrieval System) which collects bioinformatic data. This integration possibility opens an interesting opportunity to deliver data resources outside of DMS scope.

The possibility of creating a separate access policy for the data stored in DMS gives the ability to create different polices which satisfy local authorities.

The DMS modules use a well defined interface for internal cooperation based upon the SOAP communication protocol. This approach enables effortless consolidation of additional modules compatible with the DMS interface regardless of the implementation language.

4. DMS architecture

DMS considered as an internal service supporting grid computing. The main task of this system is storing and managing the computational data. One of the additional functions is proxy to other databases and data accessed from the Internet. The next function is mirroring the whole or a part of the external database. DMS consists of three logical layers. The highest layer of the DMS is a metadata repository that keeps information about the data managed by the system. The second layer consists of the Data Broker and the Mirror & Proxy. The Data Broker is an interface between the external client and DMS. The client (e.g. external service) can be a grid broker or special software DMS client e.g. computational web portal. The Data Broker can also arrange several replicas of data. The mirror and Proxy is responsible for accessing and mirroring external databases. The main purpose of this module is to grant access to various external objects in a uniform way. The lowest layer is the Data Storage. This layer is responsible for the physical storage of data. This module can store data in several ways: in computer filesystems, in databases or on tape storage systems.

The whole system uses the "web services" approach to co-operate between all modules and other components of the grid. Great emphasis was placed on the design system according to other data management systems and the proposed standards from the international bodies. The described structure is shown in the figure below.

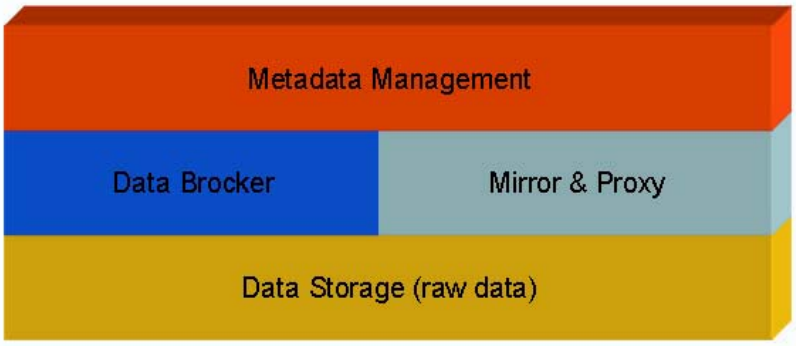

Fig. 4.1

- **The Metadata Management and repository**

The Metadata Management module (MD) is one of the most important modules. You can think of the repository as a place where metadata are stored. The Metadata Management module is responsible for storing and manipulating metadata. The data are described with attributes and access rules. In fact the metadata are stored in an object database in a *property=value* format. This format can be defined by user or chosen from the predefined formats like the Dublin Core [14] (DC) standard. We decided to run the MD module as a single instance service. This decision was made because of complexity with handling metadata information which is stored in the MD module. Running multiple instances would generate issues, which concern problems with data consistency and reliability. Consistency and reliability of metadata can be provided by a replication mechanism in the underlying database system.

- **The Data Broker and Mirror & Proxy module**

The Data Broker (DBR) is an access point of the DMS system. The key functions of this module are: serving the client requests with authorization and managing of replicas in DMS. DBR module is the access point for all other services or users requesting data operations. This module can be run in multiple instances to assure reliability and good scalability. All kinds of requests addressed to the DMS system flows through the DBR module. The system can use multi instances of this module. This approach provides the system with efficient data processing. Because of a need for client authorization procedures in the DBR module it is possible to create local system access polices for users. Polices are managed by organizations which runs the DBR module on their resources. The DBR module manages data replicas that are stored in Data Storage modules. The main function of the Proxy module is to relay communication between a client and shared resources in the Internet. This module caches frequently accessed resources and offers uniform interface. This module stores the cashed data with special attributes in the Data Storage module. The basic functions of the Mirror module are to care about the consistency of data and mirror the earlier defined set of data. This module can use the remote resources through the proxy module.

- **The Data Storage**

The Data Storage module enables access to physical data. The data are arranged in data containers and can be stored on all media types and accessed by uniform interface. The data can be organized as files on generic filesystems, BLOBs in databases or files on data tapes. In the distributed scheme Data Storage modules can store data in one or more media. The most important functions of this module are: reading and writing data, providing

data security, sharing data via protocols: GASS [7], GridFTP [8] and FTP. The DMS system can use multiple instances of the DS module. This approach enables uninterrupted data accessibility even in case of network connection failure or system crash on which the DS module operates. This module is also capable of finding optimal DS module for client connection.

The general architecture of the DMS system is shown in fig. 4.2

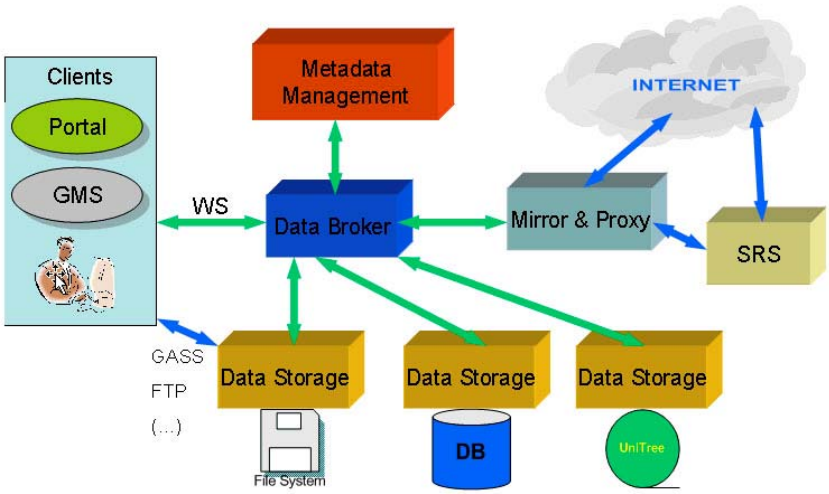

Fig. 4.2

5. Managing of the data files' transferring in the DMS

The main part of the DMS, which plays a significant role during the process of the files' transferring, is the Repository Catalog (RC). This module generally acts as the information service provider that can be queried to discover the physical locations of logical files.

Additionally, the RC component must interact with the Data Containers to estimate the optimal location for storing or retrieving data.

The metadata catalog contains semantic information that describes the location and content of data files. Users or applications can query this Metadata Catalog to identify files with the desired attributes or to obtain the physical file's location.

Storing and retrieving data in the DMS

Unique functionality implemented in the DMS is handling the date transfer from and to the DMS. The uniform mapping between the logical file's

instances and their physical locations is done by keeping all the required information in the DMS, which are the following (according to [12]):

LFN – *logical file name.* This attribute is globally unique in the DMS for a given dataset by using a unique number generated by the repository.

SFN – *site file name*. The site file name consists of a "machine:port/directory/LFN".

SURL – *site URL*. This is a "site URL" which consists of "protocol://SFN".

TFN – *transfer file name*. This is the "transfer" file name of the actual physical location of a file that needs to be transferred. It has a format similar to an SFN

TURL – *transfer URL*. This is the "transfer URL" that DMS returns to a client for the client to "get" or "put" a file in that location. It consists of "protocol://TFN", where the protocol must be a specific transfer protocol selected by client from the list of protocols supported by the DMS. If the physical storage location matches the one provided by the SURL, then only the protocol is replaced in the TURL.

In order to discuss the process of retrieving data in the DMS we have illustrated this by an example (see Fig.5.1).

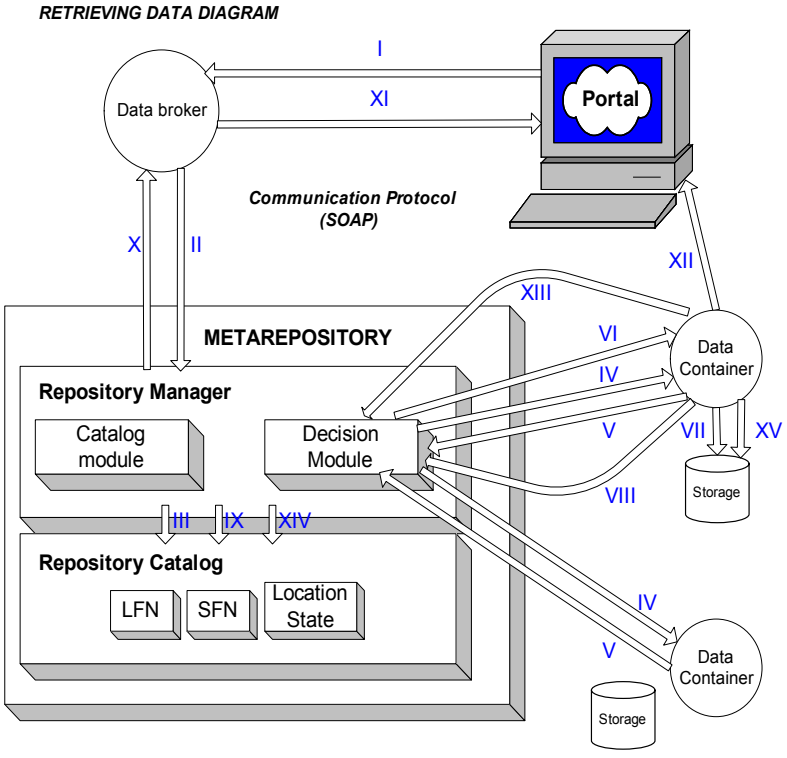

Fig. 5.1

We assume that a client application wants to access a data file and it possesses its logical name only. Let us make another assumption that the client will access data via FTP protocol. The client program sends a logical file name to the Data Broker in order to receive the site where the corresponding physical file is stored (I). The Data Broker accepts this request and first it asks the authorization module if the user has the appropriate rights to perform this operation. The successful authorization is required to carry on this request execution; otherwise the request is rejected.

In the next step DB creates the query to the Repository Manager (RM), which can be presented in a readable form: "Get the URL for the file identified by LFN ready to get via FTP protocol" (II). The SOAP envelope is formed and passed on to the RM.

Now the RM has to determine the optimal location and it performs several steps to accomplish the job:

Enter the critical section. This prevents deadlock of the concurrent process

Retrieve all the SFN of the given logical file's name from the metadata (III)

Query the remote Data Containers if the analyzed file is ready to download to determine the optimal file location for the given request (IV, V)

On the basis of all the collected data (described in the previous steps) RM makes a decision from where the file will be downloaded, enabling them to read access on the appropriate DC (VI).

The selected Data Container blocks the physical file from deleting (VII), return the site URL of the chosen data file and the RM change the meta-file's location state to the WRITE_LOCK mode for that file (IX).

RM creates a TURL and leaves the critical section

This TURL is sent to the Data Broker as response (X) and it is returned to the client application (XI) to realize transfer of data (XII).

Finally the Data Container, which was chosen as a source of the transfer data file, watches the transfer course. When it finishes, DC informs the RM (XIII) about that situation. Now the RM can release the WRITE_LOCK in the Repository Catalog (XIV) and on the Data Container module (XV).

In order to discuss the process of storing data in the DMS in a more detailed way we have illustrate this by an example (see Fig.5.2).

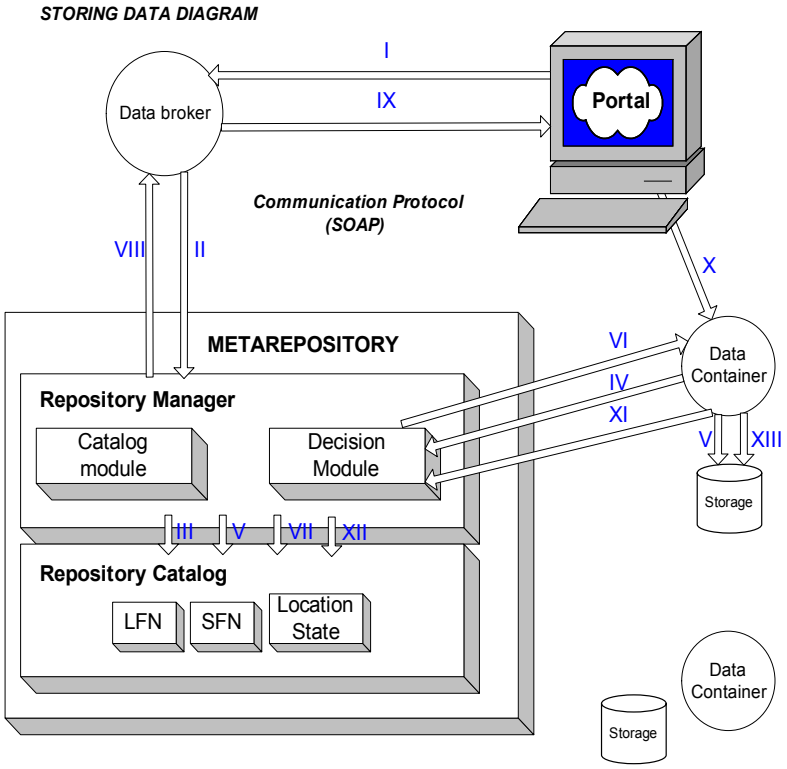

Fig. 5.2

Let us have a look at data storing by means of modules of the DMS. We assume a client application wants to store a set of data (a data file) under a logical name using GASS protocol for transfer. The client program announces the Data Broker that it wants to put the data file under the DMS control having a logical file name (which comes from the user decision) (I). When the Data Broker gets that kind of request, it has to check the user permission to operate the stored data. To hold this, it first asks the authorization module, if user has the appropriate rights to perform this task. The successful authorization is required to carry on this request execution; otherwise the request is rejected.

The next step belongs to the DB. It sends the save-data request to the Repository Manager (RM) (II). That kind of request causes the RM to create a record about file in the metarepository (III). This operation completes with getting the logical file name for a new user file (name which is an internal system name of that file). In the next step the RM tries to find the location for a new file. Having additional information about the expected size and preferred protocol to realize the transfer, the RM has to determine the optimal location and it performs several steps to accomplish the whole job:

Choose list of appropriate Data Containers for storing data file of the given size under a given logical file's name according to the information about the accessible Data Containers collected in the metadata

Query the remote Data Containers to determine if it is able to get the data file for the given request

On the basis of all the collected data (described in the previous steps) RM make a decision where the file will be uploaded.

The selected Data Container is asked to reserve resource space for the data file (IV) and it allocates space for data in the storage resource controlled by it (V)

The SFN (site file name) can be constructed now, and it is stored in appropriate record in the Repository Catalog (V)

The DC has been determined as well, so the site URL (transfer file name) can now be generated (VI)

RM adds the meta-file's record and location for a new data file with state set to the IN_TRANSFER mode (VII)

This site URL (TFN) is sent to the Data Broker as a response (VIII) and it get to the client application (IX).

After these tasks the client application gets the transfer URL, which can be used by it to realize the real transfer. The data file will be ready to use when the file transfer to the DC is finished. The Data Container that got the new file (according to the previous reservation) reacts to the transfer finish (or fail if occurs) and informs the RM about state of this operation. The task can be finished with setting the state of the new file to the READY (if transfer succeed) or the record about the new file (the transfer of which failed) can be removed from the repository.

6. Future work

The PROGRESS project is currently under development. Time frame for the DMS which is a part of it is in May 2003. The most important functionality has already been implemented. Next plans cover the DMS adoption to OGSA specification [15]. This enables improved integration capability with other grid driven projects.

Data management system is required for grid computational environments. Data sharing and its integration is a crucial issue in heterogeneous computational clusters. Considering this problem we have come up with the idea of the DMS system. The DMS incorporates best features of highly available and scalable systems. We think it will develop towards open service for data management in grid-portal architecture.

7. Bibliography

1. GriPhyN - Grid Physics Network website. Accessed from http://www.griphyn.org/index.php
2. SDSC Storage Resource Broker website. Accessed from http://www.npaci.edu/DICE/SRB/
3. Spitfire Homepage. Accessed from http://spitfire.web.cern.ch/Spitfire/
4. Work Package 2 of the European DataGrid project website. Accessed from http://grid-data-management.web.cern.ch/grid-data-management/
5. European Union DataGrid Project: http://eu-datagrid.web.cern.ch/eu-datagrid/
6. PROGRESS website. Accessed from http://progress.psnc.pl/
7. Global Access and Secondary Storage (GASS). Accessed from http://www-fp.globus.org/gass/
8. The GridFTP Protocol and Software. Accessed from http://www-fp.globus.org/datagrid/gridftp.html
9. The DataTAG project website accessed from: http://datatag.web.cern.ch/datatag/
10. Storage Resource Management (SRM) Middleware Project website. Accessed from: http://sdm.lbl.gov/indexproj.php?ProjectID=SRM
11. Particle Physics Data Grid collaboration website. Accessed from: http://www.ppdg.net/
12. SRM Joint Functional Design - Summary of Recommendations, January 2002
13. Rychlewski, J., Weglarz, J., Starzak, S., Stroinski, M., Nakonieczny, M.: PIONIER: Polish Optical Internet. Proceedings of ISThmus 2000 Research and Development for the Information Society conference Poznan Poland (2000) 19-28
14. Dublin Core standard: http://dublincore.org/
15. Open Grid Services Architecture: http://www.globus.org/ogsa/