LHC Grid Computing Project

ARDA Prototype – Working Document

DRAFT

Document identifier:	XXX-XXX-XX-XXXX
Date:	
Authors:	
Editors:	
Document status:	**DRAFT v0.11**

Abstract: This working document is used to break down the high level services defined by ARDA to actual components and tries to define the initial set of services provided by the prototype, their interfaces, as well as the technology/systems exploited. The appendix maps these components to existing implementations coming from Alien, EDG, and VDT.

The structure and initial AliEn input is taken from Chapter 5 of Draft v0.2 of the ARDA document.

Issue	Date	Comment
0.1	08-Dec-2003	First version (E.L.)
0.2	16-Dec-2003	Added more recent AliEn description and general description of services from ARDA document (P.B.)
0.3	17-Dec-2003	Added replica management description from Peter K. (E.L.)
0.4	18-Dec-2003	Added R-GMA description from Steve F. (E.L.)
0.5	19-Dec-2003	Re-ordered sections (E.L.)
0.6	06-Jan-2004	Updated information section (SMF)
0.7	12-Jan-2004	Added description of AliEn I/O (P.B.)
0.8	12-Jan-2004	Added contributions from Francesco Prelz on the "WP1" Network Server interface and the CE functionality that is obtained via CondorG/GRAM (E.L.)
0.9	16-Feb-2004	Added initial data mgmt API from Peter K. (E.L.)
0.10	23-Feb-2004	Added more input on data mgmt from Peter K. (E.L.)
0.11	23-Feb-2004	Document re-structuring: previous work from Alien/EDG/VDT moved to appendix (E.L.)

CONTENT

1 INTRODUCTION 4

2 TERMINOLOGY 6

2.1 API DEFINITION 7
2.2 CONVENTIONS FOR DESCRIBING METHODS 7
2.3 ERROR HANDLING 7
2.4 POSIX ACLS 8
2.4.1 ACL Types 8
2.4.2 ACL Entries 8
2.4.3 Valid ACLs 9
2.4.4 Correspondence Between ACL Entries and File Permissions 9
2.4.5 Object Creation and Default ACLs 9
2.4.6 Access Check Algorithm 9
2.4.7 Acl Text Forms 10
2.4.8 Rationale 11
2.4.9 Changes to the File Utilities 11
2.4.10 Standards 11

3 GENERAL DECOMPOSITION 12

3.1 API AND GRID ACESS SERVICE 13
3.1.1 Data Services 14
3.2 AUTHENTICATION 15
3.3 AUTHORISATION SERVICE 15
3.4 INFORMATION SERVICE 15
3.5 GRID MONITORING 16
3.6 JOB MONITORING 16
3.7 JOB PROVENANCE 16
3.8 AUDITING SERVICE 16
3.9 ACCOUNTING SERVICE 16
3.10 SITE GATEKEEPER 17
3.11 COMPUTING ELEMENTS 17
3.12 STORAGE ELEMENTS 17
3.12.1 Management API 17
3.12.2 Posix I/O 18
3.13 WORKLOAD MANAGEMENT 21
3.14 DATA SERVICES 21
3.15 DATA SCHEDULING 22
3.15.1 API Methods 22
3.16 FILE CATALOGUE 27
3.16.1 Basic concepts 27
3.16.2 Functional concepts 28
3.16.3 Issues, Discussion 29
3.16.4 API Definition 29
3.17 METADATA CATALOGUE 40
3.17.1 File-based Metadata API 40
3.18 PACKAGE MANAGER 40

4 REFERENCES 41

1 INTRODUCTION

During the ARDA workshop held at CERN on Dec. 3rd/4th 2003 it was decided to use the component breakdown and its mapping to AliEn contained in Draft v0.2 of the ARDA document as the working basis for developing a concrete component description and implementation recommendation for the ARDA prototype work.

The attendees of the ARDA workshop were:

- Predrag Buncic (AliEn)
- Miron Livny (Wisconsin/VDT)
- Francesco Prelz (INFN)
- Torre Wenaus (LCG AA)
- Peter Kunszt (CERN)
- Frederic Hemmer (CERN)
- Erwin Laure (CERN)
- Steve Fisher (CLRC)

Figure 1 presents the high level design that was developed during this workshop (original) based upon the Alien design, Figure 2 presents an extract of that.

Figure 1 Original Design

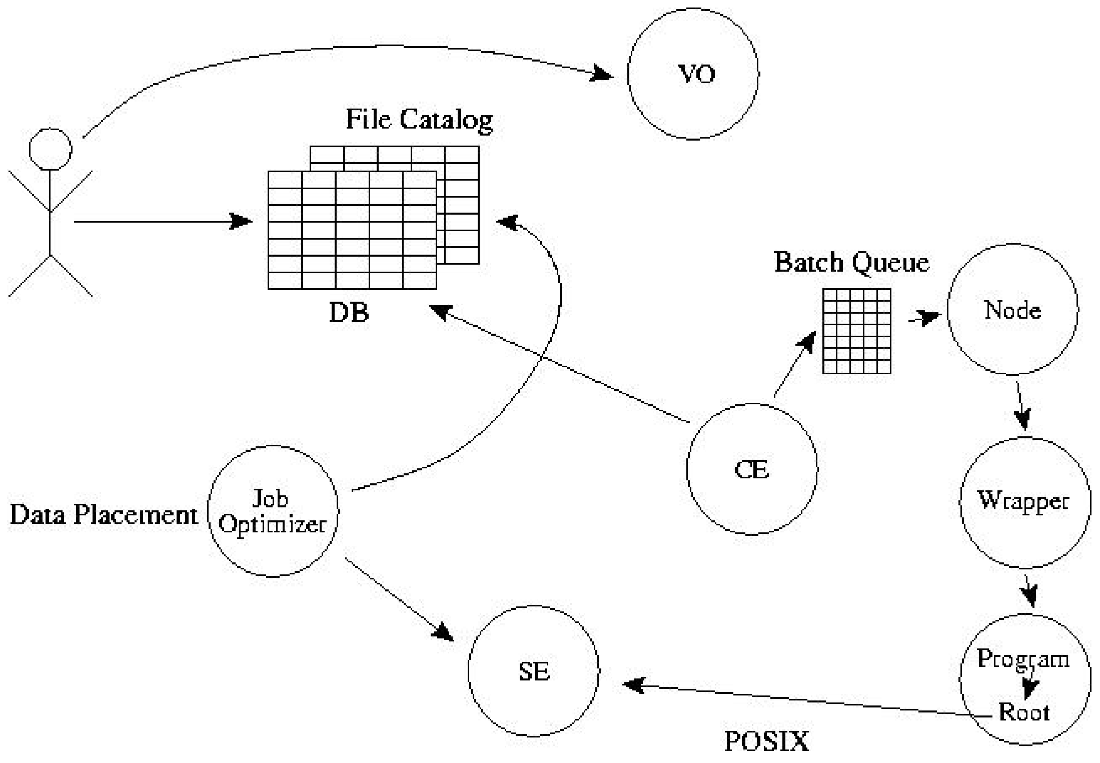

Figure 2 Transcript of Original Design

In the remainder of this document we further develop this design based on existing components from Alien, EDG, and VDT, which are described in the appendix.

2 TERMINOLOGY

The terminology listed here is a suggestion to be adapted throughout the ARDA prototype. The acronyms and names suggested below have been suggested through HEPCAL and HEPCAL-II or emerged as the established names in the Grid community in the last couple of years.

Logical File Name	LFN	Unique human-readable identifier of a Grid file.
Global Unique ID	GUID	Unique identifier by construction for a file. Think of it as an inode.
Storage Resource Manager	SRM	A service providing a management interface to mass storage.
Storage URL	SURL	URI of a file on an SRM or at generic storage. The URI has the scheme 'srm' for files managed by an SRM and 'sfn' for other files.
Site		A Grid Site is an administrative domain providing computing and storage resources.
Virtual Organization	VO	A set of Grid users characterized by common usage and access capabilities. Every Grid user belongs to at least one VO.
Storage Element	SE	An SE provides storage to the Grid users. It usually has an SRM interface, but it can also be a simple FTP server or a SAN.
Catalog		A catalog is a collection of data that is updateable and transactional.
Dataset	DS	A dataset is a read-only collection of data.

At this point the suggestion is to introduce an explicit distinction between APIs intended for the human Grid user and for the Grid middleware 'user'. Humans have very different semantic expectations of APIs than automated services, so the exposed APIs are different and have different reasons to exist.

User-Domain API	The API intended for the human user. This includes the API provided to the application programmer, i.e. all high-level application specific tools and interactive processes should be implemented using this API.
Middleware-Domain API	The middleware at this point is being defined as grid services in the same grid layer, providing services to the end user by coordinating low-level grid services and basic computing resources. The Middleware domain API of a low-level service is being used by other middleware to achieve its higher-level task.
Admin API	This is also intended for the human user, but restricted to administrators. It is often useful to explicitly specify that certain APIs are only for administrative usage.

2.1 API DEFINITION

In the API definitions below a set of needed functionalities and the way they are used in the scope of a user and administrator of a Grid Service are defined. Since this specification pertains to the usable prototype to be built, we decided to divide interface into three dimensions:

- Structures (objects) to be exposed by the API. Some of the structures must be well defined to preserve semantics throughout the proposal, some may have placeholders to indicate dependence on other services APIs.
- External API exposed to user,
- External Commands - exposed as a command shell, usable by user already in the first iteration of prototyping. Usually wrappers of the equivalent Extenal User API calls or aggregation of these.

Each of this dimensions subsequently covers following areas:

- Regular user operations
- Administrative operations
- Middleware API used usually by other services to be discussed in other documents

2.2 CONVENTIONS FOR DESCRIBING METHODS

We chose C as the language to describe the API because this is what we expect most of our users to be familiar with. It is straightforward to imagine the analogous API in java and C++ or other languages but this is the subject of a more detailed document once the API is more stable.

Each method is prefixed with `grid_` in order to distinguish them from system calls. This is a C specific notation, for java and C++ name-spacing is more appropriate.

The security model is still not fixed, so some details might change that respect. For the basic methods described here we assume that the user has already authenticated to the service and that the service can enforce the user's access rights based on that authentication.

2.3 ERROR HANDLING

The error handling is uniform throughout all methods. Each method returns an error number. The error message can be retrieved through the `grid_strerror` method.

- The error handling as described here is specific to C. It is possible to map error numbers and error messages directly into exceptions which is a more common error reporting model for java and C++.
- The error mechanism is that of the unix system calls. Neither AliEn nor EDG have had this concept in their user API, however there have been many requests by the users of EDG to provide these semantics. The advantage is that it is well-known and is a de-facto standard.

TheAPI tables in the rest of the document describe each API, its input value and possible errors. The notes contain arguments why the call was chosen to be included in the specification and how it relates to AliEn and EDG as well as known issues. The table below describes the method to retrieve the error message based on an error number.

Name	**`grid_strerror`**	
Synopsis	Retrieve string error message for a given error number.	
Fields	`int errno`	Error returned by a call

	`char *buf`	Buffer to place error message in.
Errors	`GRID_EINVAL`	Not a valid error number
	`GRID_ERANGE`	Buffer not sufficient to store error message
Notes	The behavior is identical to the unix system call strerror. The error numbers and names are prefixed as well to distinguish them from the system errors.	

2.4 POSIX ACLS

The following is taken from the Posix ACL man pages.

2.4.1 ACL Types

Every object can be thought of as having associated with it an ACL that governs the discretionary access to that object; this ACL is referred to as an access ACL. In addition, a directory may have an associated ACL that governs the initial access ACL for objects created within that directory; this ACL is referred to as a default ACL.

2.4.2 ACL Entries

An ACL consists of a set of ACL entries. An ACL entry specifies the access permissions on the associated object for an individual user or a group of users as a combination of read, write and search/execute permissions.

An ACL entry contains an entry tag type, an optional entry tag qualifier, and a set of permissions. We use the term qualifier to denote the entry tag qualifier of an ACL entry.

The qualifier denotes the identifier of a user or a group, for entries with tag types of GRID_ACL_USER or GRID_ACL_GROUP, respectively. Entries with tag types other than GRID_ACL_USER or GRID_ACL_GROUP have no defined qualifiers. The following entry tag types are defined:

GRID_ACL_USER_OBJ	The GRID_ACL_USER_OBJ entry denotes access rights for the file owner.
GRID_ACL_USER	GRID_ACL_USER entries denote access rights for users identified by the entry's qualifier.
GRID_ACL_GROUP_OBJ	The GRID_ACL_GROUP_OBJ entry denotes access rights for the file group.
GRID_ACL_GROUP	GRID_ACL_USER entries denote access rights for groups identified by the entry's qualifier.
GRID_ACL_MASK	The GRID_ACL_MASK entry denotes the maximum access rights that can be granted by entries of type GRID_ACL_USER, GRID_ACL_GROUP_OBJ, or GRID_ACL_GROUP.
GRID_ACL_OTHER	The GRID_ACL_OTHER entry denotes access rights for processes that do not match any other entry in the ACL.

When an access check is performed, the GRID_ACL_USER_OBJ and GRID_ACL_USER entries are tested against the effective user ID. The effective group ID, as well as all supplementary group IDs are tested against the GRID_ACL_GROUP_OBJ and GRID_ACL_GROUP entries.

2.4.3 Valid ACLs

A valid ACL contains exactly one entry with each of the GRID_ACL_USER_OBJ, GRID_ACL_GROUP_OBJ, and GRID_ACL_OTHER tag types. Entries with GRID_ACL_USER and GRID_ACL_GROUP tag types may appear zero or more times in an ACL. An ACL that contains entries of GRID_ACL_USER or GRID_ACL_GROUP tag types must contain exactly one entry of the GRID_ACL_MASK tag type. If an ACL contains no entries of GRID_ACL_USER or GRID_ACL_GROUP tag types, the GRID_ACL_MASK entry is optional.

All user ID qualifiers must be unique among all entries of GRID_ACL_USER tag type, and all group IDs must be unique among all entries of GRID_ACL_GROUP tag type.

The `grid_acl_get_file()` function returns an ACL with zero ACL entries as the default ACL of a directory, if the directory is not associated with a default ACL. The `grid_acl_set_file()` function also accepts an ACL with zero ACL entries as a valid default ACL for directories, denoting that the directory shall not be associated with a default ACL. This is equivalent to using the `grid_acl_delete_def_file()` function.

2.4.4 Correspondence Between ACL Entries and File Permissions

The permissions defined by ACLs are a superset of the permissions specified by the file permission bits. The permissions defined for the file owner correspond to the permissions of the GRID_ACL_USER_OBJ entry. The permissions defined for the file group correspond to the permissions of the GRID_ACL_GROUP_OBJ entry, if the ACL has no GRID_ACL_MASK entry. If the ACL has an GRID_ACL_MASK entry, then the permissions defined for the file group correspond to the permissions of the GRID_ACL_MASK entry. The permissions defined for the other class correspond to the permissions of the GRID_ACL_OTHER_OBJ entry.

Modification of the file permission bits results in the modification of the permissions in the associated ACL entries. Modification of the permissions in the ACL entries results in the modification of the file permission bits.

2.4.5 Object Creation and Default ACLs

The access ACL of a file object is initialized when the object is created with `grid_mkdir()`, `grid_register()`, `grid_symlink()` functions. If a default ACL is associated with a directory, the default ACL of the directory is used to determine the ACL of the new object, i.e. the new object inherits the default ACL of the containing directory as its access ACL.

The new object is assigned an access ACL containing entries of tag types GRID_ACL_USER_OBJ, GRID_ACL_GROUP_OBJ, and GRID_ACL_MASK. The permissions of these entries are set to the permissions specified by the file creation mask.

2.4.6 Access Check Algorithm

A process may request read, write, or execute/search access to a file object protected by an ACL. The access check algorithm determines whether access to the object will be granted.

1. **If** the effective user ID of the process matches the user ID of the file object owner, **then if** the GRID_ACL_USER_OBJ entry contains the requested permissions, access is granted, **else** access is denied.
2. **Else if** the effective user ID of the process matches the qualifier of any entry of type GRID_ACL_USER, **then if** the matching GRID_ACL_USER entry and the GRID_ACL_MASK entry contain the requested permissions, access is granted, **else** access is denied.

3. **Else if** the effective group ID or any of the supplementary group IDs of the process match the qualifier of any entry of type GRID_ACL_GROUP, **then if** the GRID_ACL_MASK entry and any of the matching GRID_ACL_GROUP group entries contain the requested permissions, access is granted, **else** access is denied.
4. **Else if** the GRID_ACL_OTHER entry contains the requested permissions, access is granted.
5. **Else** access is denied.

2.4.7 Acl Text Forms

A long and a short text form for representing ACLs is defined. In both forms, ACL entries are represented as three colon separated fields: an ACL entry tag type, an ACL entry qualifier, and the discretionary access permissions. The first field contains one of the following entry tag type keywords:

user	A user ACL entry specifies the access granted to either the file owner (entry tag type GRID_ACL_USER_OBJ) or a specified user (entry tag type GRID_ACL_USER).
group	A group ACL entry specifies the access granted to either the file group (entry tag type GRID_ACL_GROUP_OBJ) or a specified group (entry tag type GRID_ACL_GROUP).
mask	A mask ACL entry specifies the maximum access which can be granted by any ACL entry except the user entry for the file owner and the other entry (entry tag type GRID_ACL_MASK).
other	An other ACL entry specifies the access granted to any process that does not match any user or group ACL entries (entry tag type GRID_ACL_OTHER).

The second field contains the user or group identifier of the user or group associated with the ACL entry for entries of entry tag type GRID_ACL_USER or GRID_ACL_GROUP, and is empty for all other entries. A user identifier can be a user name or a user ID number in decimal form. A group identifier can be a group name or a group ID number in decimal form. The third field contains the discretionary access permissions. The read, write and search/execute permissions are represented by the `r`, `w`, and `x` characters, in this order. Each of these characters is replaced by the `-` character to denote that a permission is absent in the ACL entry. When converting from the text form to the internal representation, permissions that are absent need not be specified.

White space is permitted at the beginning and end of each ACL entry, and immediately before and after a field separator (the colon character).

2.4.7.1 Long Text Form

The long text form contains one ACL entry per line. In addition, a number sign (#) may start a comment that extends until the end of the line. If an GRID_ACL_USER, GRID_ACL_GROUP_OBJ or GRID_ACL_GROUP ACL entry contains permissions that are not also contained in the GRID_ACL_MASK entry, the entry is followed by a number sign, the string "effective:", and the effective access permissions defined by that entry. This is an example of the long text form:

```
user::rw-
user:lisa:rw-         #effective:r—
group::r—
group:toolies:rw-     #effective:r—
mask::r—
other::r--
```

2.4.7.2 Short Text Form

The short text form is a sequence of ACL entries separated by commas, and used for input. Comments are not supported. Entry tag type keywords may either appear in their full unabbreviated form, or in their single letter abbreviated form. The abbreviation for user is `u`, the abbreviation for group is `g`, the abbreviation for mask is `m`, and the abbreviation for other is `o`. The permissions may contain at most one each of the following characters in any order: `r, w, x`. These are examples of the short text form:

```
u::rw-,u:lisa:rw-,g::r--,g:toolies:rw-,m::r--,o::r--
g:toolies:rw,u:lisa:rw,u::wr,g::r,o::r,m::r
```

2.4.8 Rationale

IEEE 1003.1e draft 17 defines Access Control Lists that include entries tag type ACL_MASK, and defines a mapping between file permission bits that is not constant. The standard working group defined this relatively complex interface in order to ensure that applications that are compliant with IEEE 1003.1 ("POSIX.1") will still function as expected on systems with ACLs. The IEEE 1003.1e draft 17 contains the rationale for choosing this interface in section B.23.

In addition to these, the mode parameter is dropped for the grid adaptation since we will enforce a default ACL at all times and in the client-service model the process ACL concept is dropped.

2.4.9 Changes to the File Utilities

On a system that supports ACLs, the file utilities ls, cp and mv change their behavior in the following way:

- For files that have a default ACL or an access ACL that contains more than the three required ACL entries, the ls utility in the long form produced by ls -l displays a plus sign (+) after the permission string.
- If the -p flag is specified, the cp utility also preserves ACLs. If this is not possible, a warning is produced.
- The mv utility always preserves ACLs. If this is not possible, a warning is produced.

2.4.10 Standards

The IEEE 1003.1e draft 17 ("POSIX.1e") document describes several security extensions to the IEEE 1003.1 standard. While the work on 1003.1e has been abandoned, many UNIX style systems implement parts of POSIX.1e draft 17, or of earlier drafts.

Linux Access Control Lists implement the full set of functions and utilities defined for Access Control Lists in POSIX.1e, and several extensions. The implementation is fully compliant with POSIX.1e draft 17; extensions are marked as such.

See also http://www.guug.de/~winni/posix.1e/download.html

3 GENERAL DECOMPOSITION

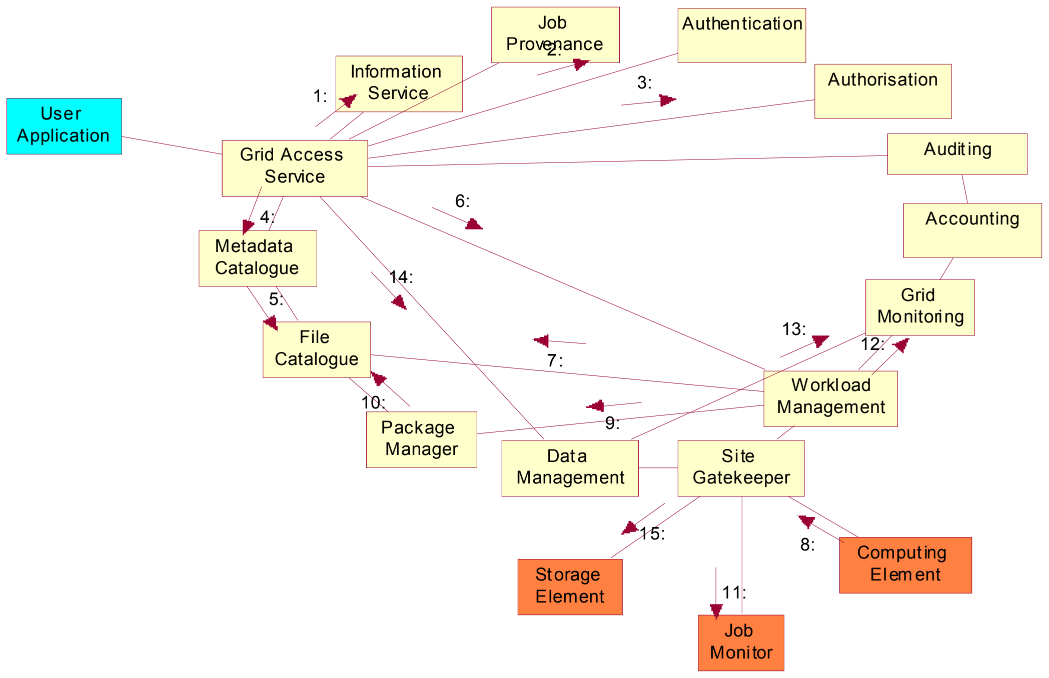

Figure 3: The interaction diagram of key Grid components for typical analysis use case

From the analysis of the AliEn architecture presented in the previous section we derive the decomposition in the following key services (as depicted in Figure 3):

- API and corresponding Grid Access Service Component
- Authentication, Authorisation, Accounting and Auditing services
- Workload and Data Management Systems
- File and Metadata Catalogues
- Information service
- Grid and Job Monitoring services
- Storage and Computing elements
- Package Manager and Job provenance service.

In the following we give descriptions of the identified services, pointing out the interfaces they provide as well as potential technologies/systems to be (re)used for the prototype implementation.

In the first phase of the prototype the focus will be on the following services:

- API and Access Service (Section 3.1)
- Authentication and Authorisation (Section 0 and 3.3)
- Information Service (Section 3.4)
- Site Gatekeeper (Section 3.10)

- Computing Element (Section 3.11)
- Storage Element (Section 3.12)
- Workload Management (Section 3.13)
- Data Scheduling (Section 3.15)
- File Catalogue (Section 3.16)
- Metadata Catalogue (Section 3.17)

More Services will follow in the future phases of the project.

3.1 API AND GRID ACESS SERVICE

An ARDA API, shown in Figure 4, would be a library of functions used for building client applications like graphical Grid analysis environments, e.g. GANGA or Grid Web portals. The same library can be used by Grid enabled application frameworks to access the functionality of the Grid services discussed in this document. The API is used also to access files available on the Grid as well as to put user's files onto the Grid. By files available on the Grid we understand those stored on one or more Storage Elements and registered in the File Catalogue or replica location service.

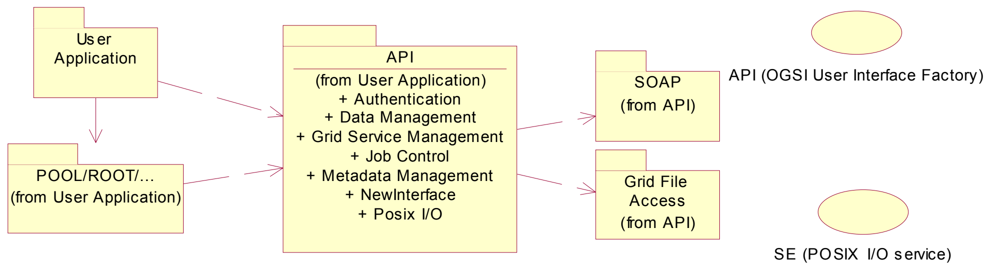

Figure 4: Grid API for user and Grid interactions

The Grid Access Service (GAS) is an example Service Component, and represents the user entry point to a set of core ARDA services. When a user starts a Grid session, he establishes a connection with an instance of the GAS created by the GAS Factory for the purpose of this session. The sequence of interactions is illustrated in Figure 5. While its creation the user is authenticated and his rights for various Grid operations are checked against the Authorisation Service. Thus the GAS is a stateful service that keeps the user credentials and authorisation information. Many of the User Interface API functions are simply delegated to the methods of the GAS. In turn many of the GAS functions are delegated to the appropriate service.

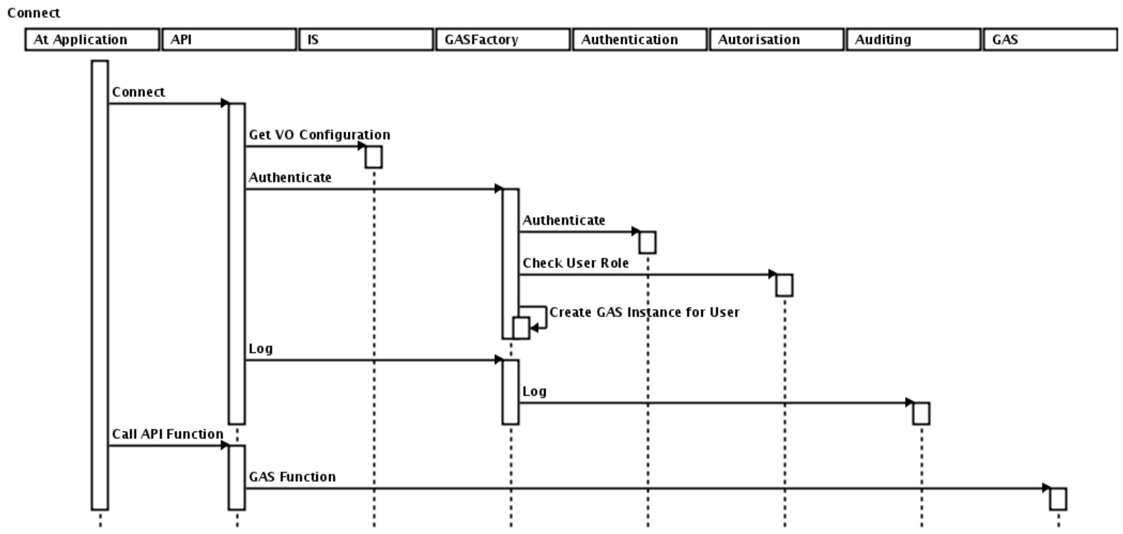

Figure 5: The sequence of interactions betweens ARDA services while application initiates connection to the Grid

3.1.1 Data Services

As a part of the ARDA API, the Metadata Catalog, SE, Data Transfer and File Catalogue AP library would expose functions used for building client applications including graphical or interactive Grid analysis environments.

The Grid Access Service (GAS) is an example Service Component, and represents the user entry point to a set of core ARDA services, exposing the API directly or indirectly.

Direct exposure is through client API bindings through java, C, C++ and scripting languages. Indirect exposure is through a unix shell-like interface or eventually through graphical user interfaces, making use of the underlying API.

In the table below there is a list of proposed commands to be implemented in the grid shell [need to specify detailed semantics and error codes].

Command	**Description**
ls	List the contents of a directory, including metadata of files (through extra options). There should be additional arguments to give the offset and the number of files to return (for scrolling purposes). Should be usable for all kinds of directories (virtual, logical) as well as replicas.
mkdir	Create a new directory
rmdir	Remove an (empty) directory
getfacl	List the ACLs applying to a file or directory
setfacl	Set the ACLs of a file or directory
whereis	Find the location of a file
tree	Print directory tree starting from a given directory. Again give offset and number of entries to return.
cp	Copy files

mv	Rename files
rm	Delete files and replicas
cd	Change directory
pwd	Print current working directory
touch	Create an empty LFN
complete	complete the given path (useful for shell completion)
locate	Find a file in the catalog

3.2 AUTHENTICATION

The Authentication Service is responsible for checking user's credentials. It can support different authentication mechanisms. It collaborates with the Information Service in order to establish the user identity.

3.3 AUTHORISATION SERVICE

Authorization Service provides information about the rights of an authenticated user to perform various operations on the Grid.

3.4 INFORMATION SERVICE

The information service is a vital component of any grid. Most services will make use of it either as publishers or consumers of information or both. Some services may simply add behavior to the information service but more commonly they will merely use the information service. These services may choose to hide the underlying information service, but this has the great disadvantage that it is then hard to combine information from the different services. It is best to start with a single well defined interface to the information system and only specialize when it proves necessary. It may be useful to provide simpler APIs for some purposes.

Any information can be monitored provided it carries a timestamp. The mechanisms to move the information around are the same. What makes monitoring systems distinctive is normally the GUIs that are provided to visualize time sequenced data and to highlight problems. These GUIs are simply clients of the information service.

Some information of interest changes rapidly and some much more slowly – however even with the slowly changing information it is often necessary to know quickly if it does change. To avoid publishing information that is only changing infrequently along with rapidly changing information is inefficient. This requires thought when designing schemas. It is better to treat the information as two or more entities with one to one relationships between them at any one time, rather than trying to bundle together slowly and rapidly changing quantities.

We can consider some of the areas where information services appear:

1. There is the "MDS like" information service used to publish information about available services.
2. The Job Provenance Service, which keeps track of the execution conditions for all the Grid jobs could be implemented on top of the information service.
3. An Auditing Service provides the mechanism for all services to report their status and error conditions. This allows Grid manager to monitor all exceptions in the system and to take corrective action. This is simply publishing information using the information service.

4. An Accounting Service could be defined to accumulate information about the use of the Grid resources by the users and groups of users. Again this does not appear to require additional functionality beyond that expected of an information service.
5. The Package Manager service gives information on the package names, versions and their locations in data repositories, usually Storage Elements. This is once more just publishing information – though perhaps a package manager might also actually manage the packages as the name indicates.
6. Application monitoring and bookkeeping. Applications being executed as part of large production runs can publish their status allowing production coordinators to keep things under control.

3.5 GRID MONITORING

The Grid Monitoring Services provides dynamic information about the status of Grid resources: computing, storage or network. This information is accumulated in the service repositories in order to have a historic view of the resource status. The clients of the service are various Grid monitoring visualisation tools as well as Workload Management Services that can optimise their scheduling decisions based on the dynamic state of the Grid and/or on historical data of the resource usage.

3.6 JOB MONITORING

Job Monitoring is a service that wraps up the running job and provides information about job status and progress. Upon request, it presents this information to other services and provides access to the job specification (JDL, etc) as well as to temporary and final files produced by the job (stdout, stderr, log files, other outputs). The Job Monitoring Service communicates with clients outside the Grid site via a Site Gatekeeper running on the gatekeeper node.. The Gatekeeper Service is either a part of the distributed Workload Management Service or an independent service.

3.7 JOB PROVENANCE

The Job Provenance Service is a specialized database to keep track of the execution conditions for all the Grid jobs. This information is used to reproduce the execution environment for the verification and debugging purposes and possibly for rerunning certain jobs. The Job Provenance Service does not contain the information used in the data queries.

To be added at a later phase of the project.

3.8 AUDITING SERVICE

An Auditing Service provides the mechanism for all services to report their status and error conditions. This allows Grid manager to monitor all exceptions in the system and to take corrective action.

To be added at a later phase of the project.

3.9 ACCOUNTING SERVICE

The Accounting Service accumulates information about the use of the Grid resources by the users and groups of users. This information serves to prepare Grid usage statistics reports. It is used also in the enhanced workload management with quotas and other policies taken into account.

To be added at a later phase of the project.

3.10 SITE GATEKEEPER

The site gatekeeper is a proxy service that routes messages to/from the worker nodes of a site. It is essential to allow worker nodes that don't have outbound connectivity to communicate with other Grid services. The site gatekeeper may also optimize the requests by e.g. message grouping or suppression of redundant requests.

3.11 COMPUTING ELEMENTS

Computing Element (CE) is a service representing a computing resource. Its interface should allow execution of a job on the underlying computing facility, access to the job status information as well as high-level job manipulation commands. The interface should also provide access to the dynamic status of the computing resource like its available capacity, load and number of waiting and running jobs. The status information should be available on per VO basis or each VO allowed to the site has its own instance of the service.

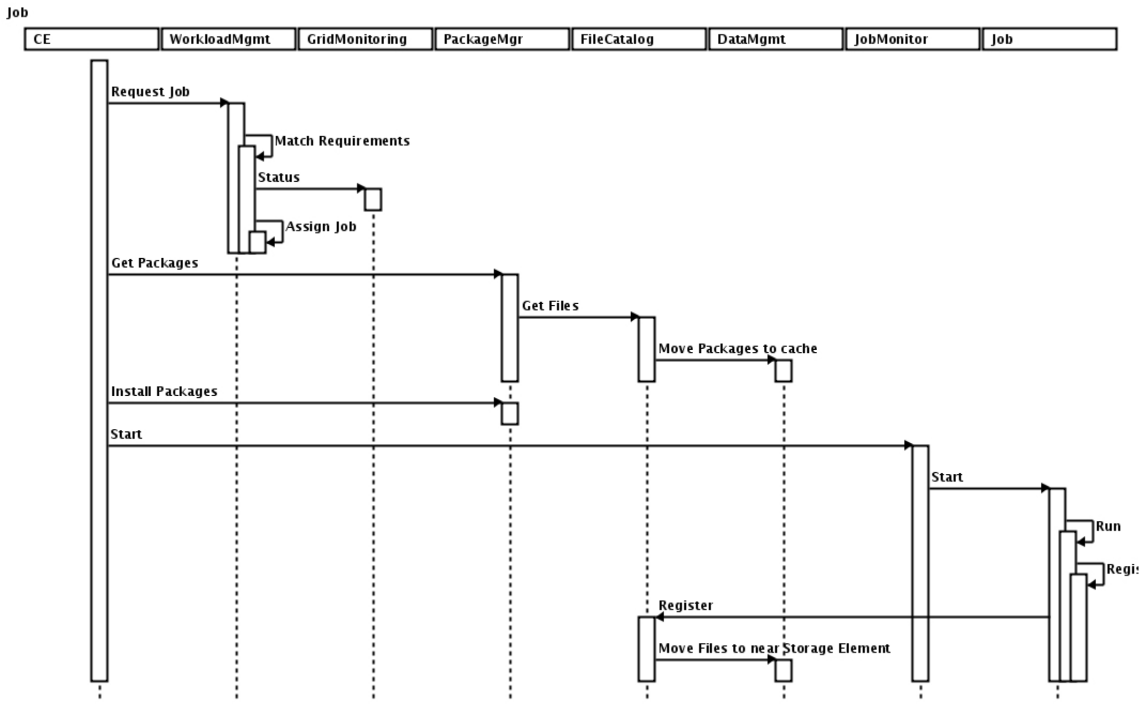

Figure 6: The sequence of interactions betweens ARDA services illustrating possible job execution model

3.12 STORAGE ELEMENTS

The Storage Element (SE) is responsible for saving/retrieving files to/from the local storage that can be a disk or a mass storage system. It manages disk space for files and maintains the cache for temporary files.

3.12.1 Management API

The storage element management interface that we propose to adopt is that of the SRM. It is described in great detail in the documents available at http://sdm.lbl.gov/srm-wg/documents.html.

This management API is intended to be used mostly by administrators and as an internal API between Grid Services.

3.12.2 Posix I/O

The interaction with the storage element should be transparent to the user through a virtual file system with posix-like semantics. Most probably we have to relax the posix API and implement only a subset, as it is done in many virtual file systems today. The final aim is to provide a grid virtual file system such that the system calls can be used for handling files. There are existing approaches to provide such a system, like AlienFS, GFAL, slashgrid, however none of them have been used and deployed in a large scale Grid yet, so an evaluation needs to be done first. Below is an API specification that provides the user with a posix-like interface with limited semantics, into which any of the above existing technologies can be factored in. For the user it should not matter which technology will be chosen as long as the functionality described below is achieved.

Name	**`grid_filedesc_init`**	
Synopsis	Initialize a grid file descriptor	
Fields	`grid_filedesc_t *fd`	File descriptor to initialize.
Errors	`GRID_ENULL`	The descriptor has not been allocated yet.
Notes	This has no correspondence in AliEn or EDG. The structure is used to store security information about the user, and session information for the Grid services.	

Name	**`grid_open`**		
Synopsis	Open a grid file		
Fields	`char* path`	Full logical file path of the file to open	
	`int flags`	The open flags. Accepted are:	
		GRID_O_RDONLY	Request file for read only.
		GRID_O_WRONLY	Request file for write only.
		GRID_O_APPEND	For write, don't overwrite but append to file
		GRID_O_CREAT	For write, if file does not exist, don't come back with an error but create the file.
		GRID_O_EXCL	If the file exists, when used with GRID_O_CREAT, an error will be returned.
	`grid_filedesc_t *fd`	The file descriptor being filled in.	
Errors	`GRID_ENULL`	The file handle structure being passed in is not initialized.	
	`GRID_ENEXIST`	Returned on read: there is no such entry in the file catalog.	

	`GRID_EUNKNOWNSURL`	The SURL which is in the catalog is not recognized by the SE. This is an inconsistency between the catalog and the actual data on the SE.
	`GRID_EACCES`	The user is not allowed to do read/write to the path.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_EISDIR`	The pathname refers to a directory and not to a file.
	`GRID_ENOTDIR`	A pathname component is in fact not a directory.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
	`GRID_EEXIST`	The pathname already exists and GRID_O_CREAT and GRID_O_EXCL were used.
	`GRID_EDENIED`	The user is not allowed to unregister.
Notes	This corresponds to the open() call in AliEn and has no direct corresponding call in EDG, where this was done in a combination of replica manager/broker info (lookup of LFN) and system setup (NFS mount). Note also that for the time being there is no read-write mode. This is due to the distributed nature of the system and a simplification in semantics, taken from AliEn.	

Name	**`grid_close`**	
Synopsis	Open a grid file	
Fields	`grid_filedesc_t *fd`	File descriptor to close.
Errors	`GRID_EBADF`	Not a valid file descriptor.
Notes	This corresponds to the close() call in AliEn and has no direct corresponding call in EDG.	

Name	**`grid_read`**	
Synopsis	Read from a grid file. Reads are sequential.	
Fields	`grid_filedesc_t *fd`	File descriptor. The file has to be opened for read.
	`const void* buf`	The buffer to be read to.
	`size_t count`	The amount of bytes to be read.
	`size_t *count`	The amount of bytes actually read.
Errors	`GRID_EBADF`	Not a valid file descriptor.

	`GRID_EINVAL`	The file descriptor given is attached to a file that cannot be written.
	`GRID_EPIPE`	While writing, the connection to the service hosting the file was lost.
Notes	This corresponds to the read() call in AliEn and has no direct corresponding call in EDG.	

Name	**`grid_write`**	
Synopsis	Write into a grid file. Writes are sequential.	
Fields	`grid_filedesc_t *fd`	File descriptor. The file has to be opened for write.
	`const void* buf`	The buffer to be written.
	`size_t count`	The amount of bytes to be written.
	`size_t *count`	The amount of bytes actually written.
Errors	`GRID_EBADF`	Not a valid file descriptor.
	`GRID_EINVAL`	The file descriptor given is attached to a file that cannot be written.
	`GRID_EPIPE`	While writing, the connection to the service hosting the file was lost.
Notes	This corresponds to the write() call in AliEn and has no direct corresponding call in EDG.	

Name	**`grid_fseek`**	
Synopsis	Position the read stream. Does not work for writes, works only forward for reads.	
Fields	`grid_filedesc_t *fd`	File descriptor. The file has to be opened for read.
	`size_t count`	The amount of bytes to skipped.
Errors	`GRID_EBADF`	Not a valid file descriptor.
	`GRID_EINVAL`	The file descriptor given is attached to a file that cannot be read (ie file has been opened for write).
	`GRID_EPIPE`	While seeking the connection to the service hosting the file was lost.
Notes	This has no corresponding call in AliEn and has no direct corresponding call in EDG.	

Name	**`grid_stat`**	
Synopsis	Return the status of a grid file based on its logical file name.	
Fields	`char *filename`	File descriptor. The file has to be opened for write.
	`struct grid_fstat_t* statbuf`	The filestat structure to be filled. It is described below.
Errors	`GRID_ENEXIST`	There is no such entry in the file catalog.
	`GRID_EACCES`	The user is not allowed to do read this information.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOTDIR`	A pathname component is in fact not a directory.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
Notes	This corresponds to the stat() call in AliEn and has no direct corresponding call in EDG.	

Note: All of the file system commands manipulating directories are described under the File Catalog component.

3.13 WORKLOAD MANAGEMENT

The Workload Management Service (WMS) receives the workload instructions from the users in the form of jobs. It is responsible for selecting the appropriate Computing Elements (CE) where the job can be run. If no CE is able to run the job, some preparatory actions can be undertaken, e.g. bringing some of the input data to a near SE. The WMS can modify the job descriptions, e.g. generate subjobs in order to optimise the overall job execution. The WMS assigns an identifier to the accepted jobs that can be used later to interrogate the job status. The WMS provides accounting information upon the job execution to the Accounting Service and can be implemented as a compact or a distributed service, i.e. having internal distributed components

3.14 DATA SERVICES

This section gives an overview of all data related services in ARDA. The main services that relate to file access in ARDA are the File Catalog, Data Management and Storage Element. Metadata Catalogs are mostly also in this area, although this depends on how ‘metadata’ is defined. We will define it through the specification in great detail below. Also of relevance are the Authentication and Authorization as well as the Access Service components. And all services need to provide Auditing and Accounting so they are discussed here as well to a limited extent. The Package Manager might make use of the other service components to achieve its task.

The granularity of the data is on the file level. The idea is to give the user the illusion of a global file system (which is specific to his VO). There may be a client application that can look like a shell (as in AliEn) which can seamlessly navigate in this virtual file system, listing files, changing directories, etc.

To read and write files is possible through a posix-compliant I/O. In terms of access control, the suggestion is to adopt posix ACLs. They have well-defined semantics and are already implemented in several file systems today. The many different flavours of mass storage systems should be hidden behind the very same posix-compliant I/O. However, the semantics of reads and write will be affected by having an MSS backend: there may be substantial latencies for reads and many more failure modes for write, so the number of errors and messages is larger than for a conventional file system. The components in Figure 3 covering these issues are the Storage Element (providing the MSS backend abstraction, virtual file system), File Catalog (providing the global logical file system view for the user), and of course the security components (Authentication, Authorization).

In a distributed environment, there will be many replicas (managed copies) of the user's files stored at different physical locations. The user does not necessarily needs to be aware of this fact, however the capabilities for controlling the replica placement need to be available. This is covered by the Data scheduling component which provides file placement capabilities.

The way the files are found and selected is not necessarily by name but by attributes, and this is where the Metadata Catalog component comes into play. However, this can be very application (and VO) specific, so the interface we specify in this document is a very simple one that can be implemented on top of existing application metadata services such that they can be used from within the grid environment. It is this mechanism that enables the concept of virtual datasets as well

3.15 DATA SCHEDULING

The Data scheduling component as defined in the ARDA RTAG document has two major components: The Data Transfer service and the Data Placement service. The transfer service responds to user requests to transfer files between two SEs. The placement service is an autonomous service that will initiate transfer (i.e. replication of files) based on access patterns and job placement considerations.

The Data Transfer service maintains and manages a queue of pending transfers across its allocated bandwidth at a given site. A user can request the replication of a given file from one site to another if he knows the file to exist at a given source and has proper access rights at the given destination. However, if the source is not known, the Data Placement service may find the 'best' source based on network monitoring to be used.

3.15.1 API Methods

The transfer scheduling and replication methods are described in detail below. The transfer scheduling is based on the AliEn transfer services described below. The replication APIs are based on the EDG replica manager.Data Transfer service

3.15.1.1 Data Transfer service

This does not include registration in the catalog. Each data transfer service is associated with a site. Just like the SE, the data transfer service should be able to serve more than one virtual organization. The concept is that the data transfer service manages transfers to and from a given site. The benefit of such a service is that identical file transfers are not initiated unnecessarily and that it provides some implicit network congestion control for this kind of operations.

Name	**`grid_schedule_transfer`**	
Synopsis	Put a new transfer request on the file transfer queue. This is a nonblocking call.	
Fields	`char * sourceURL`	The source file name. This needs to be a valid SURL recognized by an SE or a http, ftp or gsiftp URL.

	`char * destURL`	The destination file name. This also is a SURL recognized by an SE or a ftp or gsiftp URL.
	`grid_stid_t *id`	The scheduled transfer id returned.
Errors	`GRID_EBADSURL`	Not a valid SURL for either source or dest.
	`GRID_EACCESS`	The user is not allowed to write / read the file.
Notes		

Name	**`grid_cancel_transfer`**	
Synopsis	Cancel a transfer request. This will simply remove an entry from the queue. If the transfer is already in progress, this has no effect.	
Fields	`grid_stid_t id`	The transfer id to be cancelled.
Errors	`GRID_EBADTRANSID`	Not a valid transfer id (anymore).
	`GRID_EACCESS`	The user is not allowed to cancel the transfer.
Notes		

Name	**`grid_transfer_status`**	
Synopsis	Show the status of the transfer.	
Fields	`grid_stid_t id`	The transfer id to listed.
	`int *stat`	The status. It can be GRID_ST_PENDING, GRID_ST_TRANSFERRING, GRID_ST_CANCELLED, GRID_ST_DONE.
Errors	`GRID_EBADTRANSID`	Not a valid transfer id.
	`GRID_EACCESS`	The user is not allowed to read the status.
Notes		

Name	**`grid_transfer_list`**	
Synopsis	Show the list of scheduled transfers. Only those transfers are listed that the user is allowed to see. This call can be called repeatedly to get the next entry from the stream. It will return GRID_EEOF if there are no more entries left to list.	
Fields	`grid_stid_t *id`	The transfer id of the entry.
	`char *source`	The source to be transferred from. This is a buffer which should be allocated by the user before calling the method.

	`char *dest`	The destination to transfer to. This is also a buffer, just like source.
	`Int *stat`	The status. It can be GRID_ST_PENDING, GRID_ST_TRANSFERRING, GRID_ST_CANCELLED, GRID_ST_DONE.
Errors	`GRID_EOF`	This is not really an error, it indicates that there are no more entries to be retrieved. The source and dest buffers and id, stat fields are unchanged if EOF is reached.
	`GRID_ERANGE`	The source or dest buffer is not big enough to store the entry. There is a field `GRID_NAME_MAX` that defines the maximal name length.
	`GRID_EACCESS`	The user is not allowed to list the status.
Notes		

3.15.1.2 Data Placement service

This service API enables the user to create new replica instances. It does involve registration in the catalog.

Name	**`grid_create_replica`**	
Synopsis	Create a replica at a given location. This will submit and monitor a file transfer to the file transfer service and update the file catalog accordingly upon successful replication. This is a blocking call until a replication and registration is complete.	
Fields	`char *path`	The full logical file name path of the file of which a new replica is to be created.
	`char* destination`	The destination to replicat to. This can be a fully qualified SURL or just the SE to replicate to.
Errors	`GRID_EEXIST`	There is already a replica at the given destination.
	`GRID_ENEXIST`	The path does not exist in the catalog.
	`GRID_EACCESS`	The user is not allowed to create a new replica at the given destination.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
Notes	The source to replicate from will be chosen by the service. Trivially it will choose a file from the local store if one is available, otherwise it chooses one at random. This corresponds to the createMirror call in AliEn and the replica manager replicateFile command in EDG. In EDG the source was chosen based on some networking monitoring metrics.	

Name	**`grid_put_file`**	
Synopsis	Put a new file into the grid explicitly. The source file can exist either on the local file system or is accessible through one of the http, ftp, gsiftp protocols. This is a blocking call for local files and nonblocking for other files where the transfer service will be used to perform the put. This call also has a bulk operation corresponding call, `grid_bulk_put_file`.	
Fields	`char *path`	The full logical file name path of the file that should exist from now on in the grid.
	`char *sourceURL`	The source URL. This has to be a valid url with schema file, http, ftp, gsiftp.
	`char *destSURL`	This is optional and can be left NULL, in which case the service will choose a name to put the file to on the local SE. Otherwise it tries to create the file using the given SURL.
Errors	`GRID_EEXIST`	There is already a file in the catalog with the given path.
	`GRID_ENEXIST`	The source does not exist or is not accessible.
	`GRID_EACCESS`	The user is not allowed to create a new file in the given path or the given destSURL.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
	`GRID_EINVSURL`	The SURL for destination given is invalid
	`GRID_ESURLEXIST`	The dest SURL already exists on the SE
Notes	This corresponds to the copyAndRegister call in the EDG replica catalog.	
Name	**`grid_bulk_put_file`**	
Synopsis	Put a set of new files into the grid. The set of files is defined in the XML string. It may contain also metadata that is to be added to each file as well as ACLs.	
Fields	`char *xml`	The XML string specifying the files and all attributes thereof to be put. [to be detailed]
	`int policy`	Flags to specify the failure policy. The options are: GRID_FAIL_ANY Fail for all if any of the files cannot be created. GRID_FAIL_ALL Fail only if none of the files can be put. The result string will contain eventual individual failures.
	`char *resultXML`	Report on the bulk operation. This is also an XML string, containing the pathnames and individual return codes. The return codes are listed below.

Errors	`GRID_EEXIST`	There is already a file in the catalog with the given path.
	`GRID_ENEXIST`	The source does not exist or is not accessible.
	`GRID_EACCESS`	The user is not allowed to create a new file in the given path or the given destSURL.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
	`GRID_EINVSURL`	The SURL for destination given is invalid
	`GRID_ESURLEXIST`	The dest SURL already exists on the SE
Notes	This corresponds to the bulkCopyAndRegister call in the EDG replica catalog. It is first checked whether the user has the right to create all logical names in the catalog before starting the copy operations through the file transfer service.	

Name	**`grid_delete_replica`**	
Synopsis	Remove a replica from a given location. This will submit a removal request to the SE and remove the entry from the catalog.	
Fields	`char *path`	The full logical file name path of the file of which a replica is to be removed.
	`char* replica`	The replica to remove. This can be a fully qualified SURL or just the SE to remove from.
Errors	`GRID_ENEXIST`	There is no such path in the catalog.
	`GRID_EINVAL`	There is no such replica to delete.
	`GRID_EACCESS`	The user is not allowed to create a new replica at the given destination.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
	`GRID_EINVSURL`	The SURL is invalid.
	`GRID_EFAIL`	The SURL could not be removed from the SE. This occurs if there is an inconsistency between the catalog and the SE. The catalog entry is removed!
Notes	It is first checked whether the user is allowed to perform the operation on the given SURL. The entry is removed from the catalog first, and only if that operation is successful will a removal request be made at the SE (which should not but can fail with GRID_EFAIL). If this was the last replica in the catalog, the logical entry is being kept, but it will have no actual replicas associated with it and the file size is reset to 0.	

3.16 FILE CATALOGUE

The file catalog is the starting point for file-based data management. In the Grid the user identifies her files by logical file names (LFNs). The LFN is the key by which the Grid services locate the actual replicas of the files. The replicas are identified by SURLs, i.e. each replica has its own SURL, specifying implicitly which SE needs to be contacted to extract the data. Usually, users should not have to deal with SURLs, in all their scope the only names they should need to use are LFNs. The Grid should provide the look and feel of a single file system.

To give this illusion, the Grid data management middleware has to keep track of SURL - LFN mappings in a scalable manner. The Grid File Catalog provides the logical file system view to the user, with all the functionality to group files into directories and to provide access control through posix ACLs. The other major task of the Grid File Catalog is to provide the mapping of the replicas. That part is also often referred to as replica catalog.

3.16.1 Basic concepts

We define the terms:

- **SURL Storage URL**: This is a physical instance of a file replica. Also referred to as the Physical File Name (PFN).
- **LFN Logical File Name**: A logical (human readable identifier) for a file.
- **GUID Globally unique ID**: A logical identifier guaranteed unique by construction, regardless of where it is produced.

The mapping we consider is 1 LFN to 1 GUID to many SURL. It is a requirement of the LFN that it be unique. The namespace of the LFN is a directory structure, e.g.

/grid/atlas.lhc.org/production/run/07/123456/calibration/cal/cal-table100

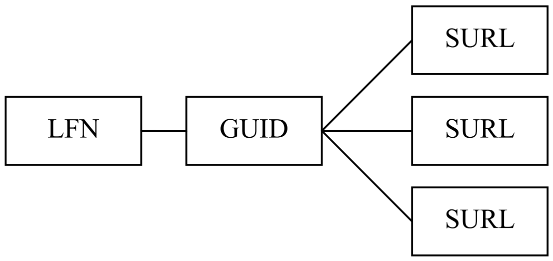

Figure 7: The mapping model. The introduction of the GUID (which is entirely kept internal) allows to distribute the catalog across the wide area and to catch accidental duplicate LFNs should they occur.

We define the LFN here to be the primary logical filename for that logical file. This full name must be unique within a VO and should be accessible to the entire VO. Secondary logical references are discussed below. **The internal GUID should never be exposed to the users, who will usually only see the human readable LFN**. The purpose of the internal GUID is to allow recovery in the case of clashes where two files are given the same LFN. An example case where this might occur is when a batch farm producing some output is disconnected from the wide area network and registers a new file (and a new LFN) in its Local File Catalog. Upon reconnection, the Local File Catalog tries to resync with the rest of the world, and finds the LFN already registered. The clash can be dealt with by some configurable policy, the easiest of which would be to mail an administrator. The GUID then gives a guaranteed-unique handle that the administrator can use to reference the file while dealing with the clash. The typical resolution would be to assign the file a different LFN. In general, the application should take reasonable steps to ensure that the LFN is unique; the process above is only for recovery purposes.

The SURL is what the storage resource (SE) uses to access the file. In the application discussed here, users should not see the SURL, only the logical filename and its directory structure. The rest of the discussion in this note pertains only to logical directories, so SURLs and physical files will not be discussed further.

For the logical side, the analogy from the UNIX file system is: The GUID is like the inode. The logical filename is a unique hard link to that inode.

3.16.2 Functional concepts

We define the concept of a *logical directory*. The directory is just as in a normal file system a list of files and other directories. It can be navigated in the same manner. The full logical filename path contains also implicitly directory structure that can be navigated in such a manner to reference all the logical files for a given VO. The grid middleware dereferences the logical filename to the SURLs, the physical instances of the file. The posix ACL semantics are also enforced through the catalog.

3.16.2.1 Metadata

In the proposed specification the Metadata Catalog module may be direcly queried through the File Catalog. The Metadata Catalog has a specific interface which returns a set of LFNs (limited to a maximal size) for a query. The schema and content of the Metadata Catalog needs to be known by the user issuing the query but is not specified by the system. We only specify the generic metadata interface and its error modes. This interface is then used by the File Catalog and is implicitly exposed through the metadata operations, which also involves virtual directories.

3.16.2.2 Logical directory

A new logical directory can be created through a simple API call (mkdir, see below). A user can copy logical files from other logical directories in to their own directory. Symbolic directory links are possible to other directories.

3.16.2.3 Read-only metadata defined virtual directory

This is a logical directory created from a metadata query (see mkdir API section). A user defines a metadata query whose output is a set of logical files that are then made accessible through the virtual directory as symlinks. Only a limited set of operations are available in such directories. We do not foresee the possibility to have subdirectories in virtual directories for example. Also, adding files to a virtual directory is only possible by refreshing the original query.

Virtual directories can be created with the grid_mkvdir command by specifying a metadata query. The result set is displayed as an enumerated set of symlinks in the directory. The query can be refreshed explicitly by issuing the grid_vdir_refresh command. The contents of the directory do not change before such a refresh operation is issued. Virtual directories have a well-defined maximal size which cannot be exceeded.

3.16.2.4 Symlinks

Similarly to symbolic links in Unix, LFN may have a nonzero number of symlinks. Symbolic links however have always be given using absolute paths. It has the same semantics as the unix filesystem symlinks, i.e. they are weak and could point to non-existing LFNs. Any operation on the target LFN does nothing to update the link, symbolic links can create cycles, etc.

3.16.2.5 HEPCAL Datasets

The HEPCAL DataSet concept maps to a logical directory which has associated metadata linked to it through the Metadata Catalog. The virtual data concept maps to the virtual directories.

3.16.3 Issues, Discussion

3.16.3.1 Security

By imposing posix ACLs on the filesystem the security semantics are rather straightforward. This should also help in avoiding concurrency issues when writing into the catalog since each user will have only limited access rights in the LFN namespace and there should be only a finite set of administrators per VO who have full access rights for all of their LFN tree. The probability of two users with the same access rights to write into the catalog in the same directory in a distributed system is therefore low.

3.16.3.2 Logical file namespace

The logical namespace needs to be unique. If we really want to exploit the filesystem semantics, it would be desirable to agree on some properties of the logical file names, as it is the case also for distributed file systems. If we take AFS as an example, it declares its root to be /afs followed by the AFS cell name. A similar idea could be applied to the Grid File Catalog namespace, i.e. start with /grid followed by the virtual organization name. Below this namespace each VO can define their own structure to prevent conflicts.

3.16.3.3 Scalability vs. Consistency

The File Catalogs that have been deployed to date are all deployed centrally and therefore are a single point of failure. The central catalog model has obviously excellent consistency properties (concurrent writes are always managed at the same place) but it does not scale to many dozens of sites. There are two possibilities to solve this issue:

1. Database Replication. The underlying database is replicated using native database replication techniques. This however means a lock-in to a vendor-specific solution. Currently commercial database vendors like Oracle provide multi-master database replication options which could be exploited, open-source solutions are not really mature yet, so this option has its limitations.
2. Lazy database synchronization exploiting the specific semantics of the File Catalog using reliable messaging to propagate the updates. Reliable messaging technologies are available (just like database technologies under point 1) both commercially and in the open source domain. The File Catalog semantics are rather simple and very specific for catalog write operations, so that every time a local write operation occurs, it can be distributed through a reliable message queue to all remote catalogs. This way one can have the same effect as under solution 1 above but without the need for vendor lock-in.

The proposal is to go with solution 2 with solution 1 as a fall-back option. Consistency might be broken in both models i.e. it is possible to register the same LFN in two remote catalogs at the same time such that a conflict will occur. The reconciliation techniques apply in both cases, however for case 2 we can be specific to the semantics of the system and exploit the uniqueness of the GUIDs.

3.16.4 API Definition

3.16.4.1 Basic types

C data types were chosen for interface definitions but with the underlying assumption that all of them could be mapped to XMLSchema 1.0 types as defined in http://www.w3.org/TR/xmlschema-2/#built-in-datatypes, including complex structures obeying this standard.

3.16.4.1.1 Logical directory methods

These methods enable interaction with the logical namespace of the catalog.

Name	`grid_readdir`
Synopsis	Return the next entry in the directory stream.

Fields	`char* directory`	Directory to list
	`char* buf`	Buffer to place next entry in.
Errors	`GRID_EBADF`	Not a valid directory
	`GRID_EOF`	This is not really an error, it indicates that there are no more entries to be retrieved or that the directory is empty. The buffer is unchanged if EOF is reached.
	`GRID_ERANGE`	The buffer is not big enough to store the entry. There is a field `GRID_NAME_MAX` that defines the maximal name length.
Notes	The behavior is similar but not identical to the posix readdir system call. This corresponds to the readdir() call in AliEn and has no correspondent in EDG.	

Name	**`grid_mkdir`**	
Synopsis	Create a new logical directory.	
Fields	`char* directory`	Full path of the directory to create
Errors	`GRID_EEXIST`	There is already an entry in the catalog with this name (either file or directory)
	`GRID_EACCES`	The parent directory does not allow the user to create a directory (see posix acl semantics below)
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
Notes	This corresponds to the MkDir() call in AliEn and has no correspondent in EDG.	

Name	**`grid_rmdir`**	
Synopsis	Delete a logical directory.	
Fields	`char* directory`	Full path of the directory to delete
Errors	`GRID_EEXIST`	There is already an entry in the catalog with this name (either file or directory)
	`GRID_EACCES`	The parent directory does not allow the user to remove a directory or directory not readable (see posix acl semantics below)
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
	`GRID_ENOTEMPTY`	The directory is not empty
	`GRID_ENOTDIR`	The path is actually not a directory

Notes	This corresponds to the RmDir() call in AliEn and has no correspondent in EDG.	

3.16.4.1.2 Virtual directory methods

Name	**`grid_mkvdir`**	
Synopsis	Create a new virtual directory.	
Fields	`char* directory`	Full path of the directory to create
	`char* queryXML`	The XML which defines the query and the query and directory metadata. It contains the directives for example about the maximal cardinality of the result, eventual offset, etc.
Errors	`GRID_EEXIST`	There is already an entry in the catalog with this name (either file or directory)
	`GRID_EACCES`	The parent directory does not allow the user to create a directory (see posix acl semantics below)
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
	`GRID_EBADQUERY`	The query has failed to create the virtual directory.
Notes	This has no corresponding call in AliEn or EDG.	

Name	**`grid_vdir_refresh`**	
Synopsis	Refresh a virtual directory.	
Fields	`char* directory`	Full path of the directory to refresh.
Errors	`GRID_ENEXIST`	There is no such entry in the catalog.
	`GRID_EACCES`	The user is not allowed to refresh the virtual directory.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
	`GRID_EBADQUERY`	The query has failed to update the virtual directory.
Notes	There is no corresponding call in AliEn or EDG.	

Name	**`grid_rmvdir`**	
Synopsis	Remove a virtual directory.	
Fields	`char* directory`	Full path of the directory to remove.
Errors	`GRID_ENEXIST`	There is no such entry in the catalog.

	GRID_EACCES	The user is not allowed to remove the virtual directory.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
Notes	There is no corresponding call in AliEn or EDG.	

3.16.4.1.3 Security methods

[to be completed in detail, including structures]

grid_acl_set_file

grid_acl_get_file

grid_acl_copy_entry, grid_acl_create_entry, grid_acl_delete_entry,
grid_acl_get_entry, grid_acl_valid

grid_acl_add_perm, grid_acl_calc_mask, grid_acl_clear_perms,
grid_acl_delete_perm, grid_acl_get_permset, grid_acl_set_permset

grid_acl_get_qualifier, grid_acl_get_tag_type, grid_acl_set_qualifier,
grid_acl_set_tag_type

grid_acl_copy_entry, grid_acl_copy_ext, grid_acl_from_text,
grid_acl_to_text, grid_acl_size

3.16.4.1.4 Logical files

Name	**grid_create**	
Synopsis	Create a new logical file name entry without actual associated data.	
Fields	char* path	Full logical file path of the file to register
Errors	GRID_EEXIST	There is already an entry in the file catalog with this name
	GRID_EACCES	The parent directory does not allow the user to write a new entry here.
	GRID_ENAMETOOLONG	The pathname is too long.
	GRID_ENOENT	A pathname component (parent) does not exist.
Notes	This does not correspond to anything in AliEn or EDG.	

Name	**grid_delete**

Synopsis	Delete logical file name entry. This will not remove the files from storage, use grid_delete_replica or grid_delete_all instead. This is purely a catalog operation.	
Fields	`char* path`	Full logical file path of the file to delete.
Errors	`GRID_ENEXIST`	There no such entry in the file catalog.
	`GRID_EACCES`	The parent directory does not allow the user to delete.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
Notes		

Name	**`grid_rename`**	
Synopsis	Rename the logical file name.	
Fields	`char* path`	Full logical file path of the file to rename
	`char* newpath`	Full logical file path of the new file name
Errors	`GRID_EEXIST`	There is already an entry in the file catalog with this name (new name)
	`GRID_ENEXIST`	There is no such path in the catalog (existing name)
	`GRID_EACCES`	The parent directory does not allow the user to remove the old entry or to write the new entry.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
Notes		

3.16.4.1.5 Symbolic links

Name	**`grid_symlink`**	
Synopsis	Create a symbolic link.	
Fields	`char* path`	Full logical file path of the new file name (i.e. the symlink)
	`char* linkpath`	Full logical file path of existing file to link to
Errors	`GRID_EEXIST`	There is already an entry in the file catalog with this name (symlink path, first arg)
	`GRID_ENEXIST`	There is no such path in the catalog (existing name to link to, second arg)
	`GRID_EACCES`	The parent directory does not allow the user to write the entry.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
Notes		

3.16.4.1.6 Replica manipulation and entry creation

These methods are all catalog-only methods and do not involve data movement. See above under 'data management' for the methods involving both registration and data movement.

Name	**grid_register**	
Synopsis	Register a file which exists on a SE.	
Fields	`char* path`	Full logical file path of the file to register
	`char* SURL`	The SURL of the
Errors	`GRID_EEXIST`	There is already an entry in the file catalog with this name
	`GRID_EACCES`	The parent directory does not allow the user to write a new entry here.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
	`GRID_EINVSURL`	The SURL given is invalid
	`GRID_ESURLNEXIST`	The SURL does not exist on the SE
	`GRID_EDENIED`	The user is not allowed to register a new replica.
Notes	This corresponds to the RegisterFile() call in AliEn and the replica manager registerFile method in EDG.	

Name	**grid_list_replicas**	
Synopsis	List all known replicas of a file.	
Fields	`char* path`	Full logical file path of the file
	`char* buf`	Buffer to place next SURL in.
Errors	`GRID_ENEXIST`	No such file.
	`GRID_EOF`	This is not really an error, it indicates that there are no more entries to be retrieved. The buffer is unchanged if EOF is reached.
	`GRID_ERANGE`	The buffer is not big enough to store the entry. There is a field `GRID_NAME_MAX` that defines the maximal name length.
	`GRID_EACCES`	The user is not allowed to read the path.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
	`GRID_EDENIED`	The user is not allowed to list replicas.
Notes	This corresponds to the replica manager listReplicas method in EDG. It has similar semantic behavior as the grid_readdir command.	

Name	**grid_unregister**

Synopsis	Un-register a file replica	
Fields	`char* path`	Full logical file path of the file modify the registrations
	`char* SURL`	The SURL of the PFN to be unregistered.
Errors	`GRID_ENEXIST`	There is no such entry in the file catalog.
	`GRID_EACCES`	The user is not allowed to do read the path.
	`GRID_ENAMETOOLONG`	The pathname is too long.
	`GRID_ENOENT`	A pathname component (parent) does not exist.
	`GRID_EINVSURL`	The SURL given is invalid
	`GRID_EDENIED`	The user is not allowed to unregister.
Notes	This corresponds to no call in AliEn and the replica manager unregisterFile method in EDG. It does not remove the LFN. If the last entry has been removed, the LFN is still not removed, it simply has no replica.	

3.16.4.2 Sessions

The methods presented here run in a context of a session. This implies that for every call *current directory* is defined, as well as *root directory*.

Many methods derive their options sets from Posix originals, many may support omnipresent grid options like asynchronous operation, publishing or others. [The details are up for discussion.]

Definitions of some of these grid options follow pseudo EBNF notation:

```
grid_option:
        grid_async_option    |
        publish_option       |
        subscribe_option
        ;

async_option :
        |
        ASYNC_FLAG timout_option return_method action
        ;

publish_option:
        |
        PUBLISH_FLAG metadata_tags
        ;

subscribe_option:
        |
        SUBSCRIBE_FLAG metadata_tags
        ;


Action:
        |
        ACTION_FLAG Method {call(Method(…)};

timout_option:
{default}|
        NUMBER SCALE     //when timeout occurs in SCALE units
        ;

return_method:
{default acknowledgment}     |
        EMAIL email_ops      |
        QUEUE queueops     |
        FILECREATION filecreatops
        ; //this also could be a list (mixture any of these)
```

Most methods in the complex and basic category can be extended to include sessions.

3.16.4.3 Complex methods

The complex operations involve the operations that do more than just atomic calls to the catalog. They operate also not just on simple types but on the structures that we define below. The methods involving session handling also are in the category of complex operations.

3.16.4.3.1 Methods involving metadata

The metadata calls are routed through to the metadata API described below in section 3.17.1.

API call	**Description**
grid_register_with_md	Register a file with a set of metadata in one go
grid_fc_add_tag	Add a tag of a certain type to a given file
grid_fc_remove_tag	Remove the tag from the file
grid_fc_show_tags	Show tags and their type associated with a file.
grid_fc_tag_exists	Ask whether a tag exists for a given file
grid_fc_set_tag_value	Set the value of a given tag of a file
grid_fc_get_tag_value	Retrieve the value of a tag of a file
grid_fc_list_files_by_tag	List all files having a given value for a tag in a directory, with offset and number of results to return
grid_fc_get_files_by_tag_search	Find all files matching a tag query, starting from a given directory, with offset and number of results to return.

3.16.4.3.2 Session-specific calls

grid_cwdir

grid_chdir

grid_pwd

3.16.4.3.3 Middleware-Domain API

The job scheduler will need to know all available replicas (SURLs) based on a (set of) LFNs. This operation has to be very efficient and has to scale very well.

[GUID-related calls – to be included]

3.16.4.4 Structures

[These need to be detailed]

Name	grid_session_t
Synopsis:	The main predefined user structure is used to store context of the user session operations, like the current directory. By definition, this

	is a singleton object per user session. Used by external programs and users to store context of a user session.	
Fields	long sessionId	Unique session ID
	string currentDirectory	Context of the session.
	string homeDirectory	Same as in u-space in unix
	long sessionLifetime	End-of-session lifetime, sessions need to be able to time out upon client failures or idleness
Notes	Sessions are always tightly coupled with the user's security object.	

Name	grid_user_t	
Synopsis:	The user's identity object based on the credentials.	
optional	string userDN	The user's distinguished name
	string[] userRoles	Roles the user has been associated with
	long validityTime	
	long creationTime	
Notes:		

Name	grid_file_t	
Synopsis:	The main predefined user structure is used to store information on a file.	
Fields	String LFN	Full path of the file
	ACL accessControl	ACL of the file
	long size	File size
	long creationTime	Creation time of file
	URI[] SURLlist	List of replicas
	URI GUID	GUID of file
	URI parent	GUID of parent directory
	FCMetadata md	Metadata on file
Notes:	The fields do not need to be present or filled for some methods that only deal with simple lookups. There are specific methods that will fill in the fields of existing FCFile objects.	

Name	grid_directory_t	
Synopsis:	Specifies the nature of a directory. It can be a virtual directory or a 'real' logical directory.	
Fields	string path	Full path of directory
	URI GUID	GUID of directory
	URI parent	GUID of parent directory
	ACL accessControl	The access control list of the directory

	boolean virtual	The virtual directory flag
	string query	Query string for virtual directories
	long maxsize	The maximal number of files in this directory
	long creationTime	Time when directory was created
	long refreshTime	Time when directory was last refreshed (query was rerun for virtual directories or last written to for logical directories)
Notes:		

Name	grid_metadata_t	
Synopsis	All metadata associated with a catalogue entry as returned through the metadata interface.	
Fields	long size	Number of key-value pairs for the entry
	URI GUID	The GUID of the LFN this metadata belongs to
	string metadata	An XML string with all the metadata filled.

3.16.4.5 Issues, Discussion

Additional types.

There are additional types and structures to be precisely defined, including simple types like timestamps but possibly complex ones as: machine data (SE), policies, tags, security objects and other types to be interchanged/reached via other services.

Bulk operations

Bulk operations have been requested by many people to increase performance and to optimise interaction with the Grid services. A simple grid_execute command might be the solution that takes an XML document containing all the operations that the user wants to perform in one go. The XML structure needs to be defined. Open issues involve behaviour on failures, transactional consistency, session management.

Metadata, HEPCAL DataSets

Hepcal only specifies dataset metadata, which would be. Do we also specify metadata on files?

Virtual directories are the result of a metadata query.

Sessions

Most methods in the complex and basic category can be extended to include sessions, which would add a session structure to each call to be tracked. This slows things down but of course increases the functionality and robustness. Should we aim for providing such sessioned versions of all calls?

3.17 METADATA CATALOGUE

The Metadata Catalogue Service contains additional information (arbitrary and extensible set of attributes) about the contents of the available files. These metadata are used for querying the Metadata Catalogue in the search for the datasets meeting the required criteria.

The metadata catalog interface is defined to serve a very well defined set of tasks. The metadata can be generic or file-based.

3.17.1 File-based Metadata API

API call	Description
grid_md_add_tag	Add a tag of a certain type to a given file
grid_md_remove_tag	Remove the tag from the file
grid_md_show_tags	Show tags and their type associated with a file.
grid_md_tag_exists	Ask whether a tag exists for a given file
grid_md_set_tag_value	Set the value of a given tag of a file
grid_md_get_tag_value	Retrieve the value of a tag of a file
grid_md_list_files_by_tag	List all files having a given value for a tag in a directory, with offset and number of results to return
grid_md_get_files_by_tag_search	Find all files matching a tag query, starting from a given directory, with offset and number of results to return.

3.18 PACKAGE MANAGER

The Package Manager Service is a specialised database for the available software packages. It keeps track of the package names, versions and their locations in data repositories, usually Storage Elements. The software package dependencies information is used by installation procedures to insure coherency between the installed packages. The service provides also the information about the lifetime of the packages that is used for the clean up of the obsolete versions installed on CE's.

The packet manager relates to the data services discussed above as such that it can make use of the LFN logical namespace and of the grid storage to keep all the packets and to make them accessible to everyone through the grid. There may be a convention to keep all packets in a well-defined namespace of the LFN tree structure.

To be added at a later phase of the project.

4 REFERENCES

[R1] D2.1 Report on Current Technology: Data Access and Mass Storage, EDG Deliverable 2.1, 20 December 2001. http://cern.ch/edg-wp2/docs/DataGrid-02-D2.1-0105-2 0.pdf

[R2] Wolfgang Hoschek, Javier Jaen- Martinez, Peter Kunszt, Ben Segal, Heinz Stockinger, Kurt Stockinger, Brian Tierney, Data Management (WP2) Architecture Report , EDG Deliverable 2.2, http://edms.cern.ch/document/332390

[R3] Data Management Workpackage, EU DataGrid: D2.2.A1 Addendum to the Data Management Architecture Report (Covering Testbed Release 2.0), EDG Deliverable 2.2.A1 http://edms.cern.ch/document/374107/addendum.pdf

[R4] Data ManagementWorkpackage, EU DataGrid: D2.5 Components and Documentation for the Final Project Release. https://edms.cern.ch/file/407063/1/finalDocumentationWP2.pdf

[R5] DataGrid WP1, Definition of Architecture, Technical Plan and Evaluation Criteria for Scheduling, Resource Management, Security and Job Description, Technical Report, EU DataGrid Project. Deliverable D1.2, September 2001. https://edms.cern.ch/file/332413/1/datagrid-01-d1.2-0112-0-3.pdf

[R6] W. Allcock, J. Bester, J. Bresnahan, A. Chernevak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke; Data Management and Transfer in High Performance Computational Grid Environments. Parallel Computing, 2002. http://www.globus.org/research/papers/dataMgmt.pdf

[R7] CASTOR web-site: http://cern.ch/castor

[R8] Gregor von Laszewski, Ian Foster, Jarek Gawor, Peter Lane: A Java Commodity Grid Kit , Concurrency and Computation: Practice and Experience, 13(8-9), 2001. http://www.globus.org/research/papers/vonLaszewski cog-cpe-final.pdf

[R9] Heinz Stockinger, Flavia Donno, Erwin Laure, Shahzad Muzaffar, Peter Kunszt, Giuseppe Andronico, and Paul Millar. Grid Data Management in Action: Experience in Running and Supporting Data Management Services in the EU DataGrid Project. In Computing in High Energy Physics (CHEP 2003), La Jolla, California, March 24-28 2003.

[R10] Diana Bosio, James Casey, Akos Frohner, Leanne Guy, Peter Kunszt, Erwin Laure, Sophie Lemaitre, Levi Lucio, Heinz Stockinger, Kurt Stockinger,William Bell, David Cameron, Gavin Mc- Cance, Paul Millar, Joni Hahkala, Niklas Karlsson, Ville Nenonen, Mika Silander, Olle Mulmo, Gian-Luca Volpato, Giuseppe Andronico, Federico DiCarlo, Livio Salconi, Andrea Domenici, Ruben Carvajal-Schiaffino, and Floriano Zini.Next-Generation EU DataGrid Data Management Services. In Computing in High Energy Physics (CHEP 2003) , La Jolla, California, March 24-28 2003.

[R11] Heinz Stockinger, Asad Samar, Shahzad Muzaffar, and Flavia Donno. Grid Data Mirroring Package (GDMP). Scientific Programming Journal - Special Issue: Grid Computing, 10(2):121-134, 2002.

[R12] SRM documents. http://sdm.lbl.gov/srm-wg/documents.html

[R13] H. B. Newman, I.C. Legrand, MonaLisa: A Distributed Monitoring Service Architecture, these proceedings, MOET001

[R14] MONARC. http://www.cern.ch/MONARC/

[R15] http://doc.in2p3.fr/bbftp/

[R16] http://asg.web.cmu.edu/sasl/