ARC middleware

The NorduGrid Collaboration

Abstract

The paper describes the existing components of ARC, discusses some of the new components, functionalities and enhancements currently under development, and outlines the architectural and technical choices that have been made to ensure scalability, stability and high performace.

1 Introduction

Computational Grids are not a novel idea. The Condor® project [1] led the way in 1988 by setting the goal of creating a computing environment with heterogeneous distributed resources. Further efforts to create a layer for secure, coordinated access to geographically distributed computing and storage resources resulted in the appearance of the Globus Toolkit® [2] and its Grid Security Infrastructure (GSI) [3] in the late 1990's. The words *Grid* and *middleware* were coined to describe the new phenomenon of the single-login access to widely distributed computing resources. Addressing the needs of the Nordic[1] research community, the NorduGrid collaboration [4] in 2002 started to develop a software solution for wide-area computing and data handling [5], making use of the by then *de facto* standard Globus Toolkit® 2. This new open source middleware is now called the Advanced Resource Connector (ARC).

Scientific and academic computing in the Nordic countries has a specific feature in a way that it is carried out by a large number of small and medium size facilities of different kind and ownership. An important requirement from the very start of ARC development has been to keep the middleware portable, compact and manageable both on the server and the client side. The client package currently occupies only 14 Megabytes, it is available for most current Linux[2] distributions and can be installed at any available location by a non-privileged user. For a computing service, only three main processes are needed:

[1] Nordic states are: Denmark, Finland, Iceland, Norway and Sweden.
[2] Linux is a trademark of Linus Torvalds.

2005

those of the file transfer service (GridFTP, see Section 3.1), the Grid Manager (see Section 3.3) and the Local Information Service (see Section 3.6.1).

Recently, ARC development and deployment has been funded by the Nordic Natural Science Research Council (NOS-N) and the aim is to provide a foundation for a large-scale, multi-disciplinary Nordic Data Grid Facility [6]. ARC is being used as the middleware of choice for this facility and for many other Grid infrastructures, such as the Estonian Grid [7], Swegrid [8], DCGC [9], Swiss ATLAS Grid [10] and others. The middleware supports production-level systems being in a non-stop operation since 2002 [11], and has proven its record in demanding High Energy Physics computing tasks [12,13], being the first ever middleware to provide services to this kind of a massive world-wide data processing effort.

This paper gives a comprehensive technical overview of the ARC middleware, its architecture and components — the existing core ones as well as the recently introduced and beta-quality services.

The rest of the paper is organised as follows. In Section 2, the basic architecture of ARC is presented. Section 3 discusses various services and Section 4 describes the client components and user interfaces. Sections 5 and 6 are more development oriented, discussing the developer interfaces and libraries, recent additions to ARC, and software dependencies and packaging, respectively. A summary is provided in Section 7.

2 Architecture

The architecture and design of the ARC middleware has been driven by the requirements and specifics of the user community that initiated the project — the Nordic ATLAS [14] groups. The basic set of high-level requirements is not different from those of any other community, be it users, resource owners or software developers: the system must be performant, reliable, scalable, secure and simple. However, the environment that led to creation of ARC has some specifics that can be outlined as follows:

- Excellent academic network (NORDUNet [15]), providing a solid basis for deployment of wide area computing architectures based on Grid technologies;
- A multitude of computing resources of various capacity, origin and ownership, effectively excluding a possibility for a centralized, uniform facility;
- Prevalence of trivially parallel, serial batch tasks, requiring Linux flavours of operating system;
- Necessity to operate with large amounts (Terabytes) of data stored in files

and accessed by the tasks via POSIX calls.

Customers' requirements impose strict rules for the middleware development. Independently of the nature of the customer — an end-user or a resource owner — the guiding principles are the following:

(1) No single point of failure, no bottlenecks.
(2) The Grid is self-organizing with no need for centralized management.
(3) The Grid is robust and fault-tolerant, capable of providing stable round-the-clock services for years.
(4) Grid tools and utilities are non-intrusive, have small footprint, do not require special underlying system configuration and are easily portable.
(5) No extra manpower is needed to maintain and utilize the Grid layer.
(6) Tools and utilities respect local resource owner policies, in particular, security-related ones.

ARC developers take a conscious approach of creating a solution that addresses the users' needs while meeting the resource owners' strict demands. This requires a delicate balance between the users' wishes to get a transparent, single-login access to as much resources as possible world-wide, and the system administrators' needs to keep unwanted customers and expensive or insecure services at bay. Simplicity is the key word: applied correctly, it ensures robustness, stability, performance and ease of control. Development of the ARC middleware follows the philosophy "start with something simple that works for users and add functionality gradually". This allows ARC to provide customers with a working setup on very early stages, and to define the priorities of adding functionality and enhancements based on the feed-back from the users.

Middleware development was preceded by an extended period of try-outs of by then available solutions and collecting feed-back from the users. This process identified a set of more specific user requirements, described in the following Section.

2.1 Requirements

Perhaps the most important requirement that affects ARC architecture stems from the fact that most user tasks deal with massive data processing. A typical job expects that a certain set of input data is available via a POSIX open call. Moreover, such a job produces another set of data as a result of its activity. These input and output data might well reach several Gigabytes in size. The most challenging task of the Grid is thus not just to find free cycles for a job, but to ensure that the necessary input is staged in for it, and all the output is properly staged out. To date, ARC is the only Grid solution which adequately addresses this demand.

Another important requirement is that of a simple client that can be easily installed at any location by any user. Such a client must provide all the functionality necessary to work in the Grid environment and yet remain small in size and simple in installation. The functionality includes, quite obviously, Grid job submission, monitoring and manipulation, possibility to declare and submit batches of jobs with minimal overhead, and means to work with the data located on the Grid. The client must come primarily as a lightweight command line interface (CLI), with an optional extension to a GUI or a portal. The requirement of CLI portability and non-root installation procedure practically implies its distribution as a binary archived package (*tar*, *gzip*). A user must be able to use clients installed in different workstations interchangeably, without the necessity to instantiate them as permanently active services. Naturally, configuration process must be reduced to an absolute minimum, with the default configuration covering most of the use cases. Last, but not least, the clients must remain functional in a firewalled environment.

There is another perspective on the Grid: that of the resource owners. The idea of providing rather expensive and often sensitive services to a set of unknown users worldwide is not particularly appealing to the system administrators, facility owners and funding bodies, whose primary task is to serve local user communities. A set of strict agreements defining a limited user base and specific services results effectively in a limited and a highly regulated set of dedicated computing resources, only remotely resembling the World Wide Grid idea. NorduGrid's approach is to provide resource owners with a non-invasive and yet secure set of tools, requiring minimal installation and maintenace efforts while preserving the existing structures and respecting local usage policies.

Each resource owner that would be willing to share some resources on the Grid, has a specific set of requirements to be met by the middleware. The most common requests are:

- The solution must be firewall-friendly (e.g., minimize number of necessary connections).
- Services must have easy crash recovery procedures.
- Allowed Grid-specific resource usage (load, disk space) should be adjustable and manageable.
- Accounting possibilities must be foreseen.
- Services must be able to run using a non-privileged account.
- Single configuration procedure for site-specific services is highly desirable.

The ARC architecture is carefully planned and designed to satisfy the above mentioned needs of end-users and resource providers. The major effort is put to make the resulting Grid system stable by design. This is achieved by identifying three mandatory components (see Figure 1):

(1) The *Computing Service*, implemented as a GridFTP-*Grid Manager* pair of services (see Sections 3.1 and 3.3). The Grid Manager (GM) is the "heart" of the Grid, instantiated at each computing resource's (usually a cluster of computers) front-end as a new service. It serves as a gateway to the computing resource through a GridFTP channel. GM provides an interface to the local resource management system, facilitates job manipulation, data management, allows for accounting and other essential funtions.

(2) The Information System (see Section 3.6) — serves as a "nervous system" of the Grid. It is realized as a hierarchical distributed database: the information is produced and stored at each service (*computing, storage*), while the distributed system of *Index Services* maintains the list of known services.

(3) The *Brokering Client* (see Section 4.2.1) — the "brain" of the Grid, which is deployed as a client part in as many instances as users need. It is enabled with powerful brokering capabilities, being able to distribute the workload across the Grid. It also provides client functionality for all the Grid services, and yet is required to be lightweight and installable by any user in an arbitrary location in a matter of few minutes.

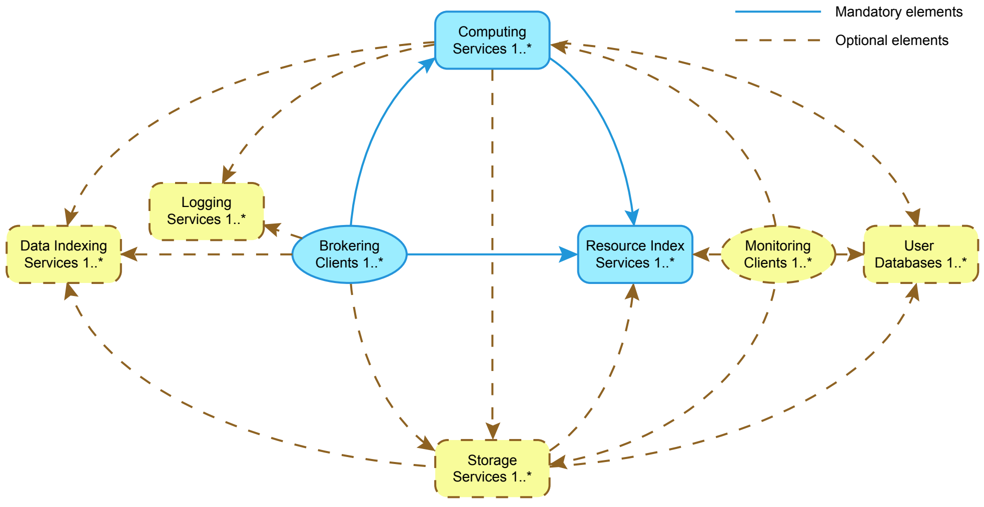

Fig. 1. Components of the ARC architecture.

In this scheme, the Grid is defined as a set of resources registering to the common information system. Grid users are those who are authorized to use at least one of the Grid resources by the means of Grid tools. Services and users must be properly certified by trusted agencies (Section 3.7). Users interact with the Grid via their personal clients, which interpret the tasks, query the Information System whenever necessary, discover suitable resources, and forward task requests to the appropriate ones, along with the user's proxy and other necessary data. If the task consists of job execution, it is submitted to a computing resource, where the Grid Manager interprets the job description, prepares the requested environment, stages in the necessary data, and submits the job to the local resource management system (LRMS). Grid jobs within the ARC system possess a dedicated area on the computing resource, called the *session directory*, which effectively implements limited sandboxing for each job. Location of each session directory is a valid URL and serves as the unique Job Identifier. In most configurations this guarantees that the entire contents of the session directory is available to the authorised persons during the lifetime of the Grid job. Job states are reported in the Information System. Users can monitor the job status and manipulate the jobs with the help of the client tools, that fetch the data from the Information System and forward the necessary instructions to the Grid Manager. The Grid Manager takes care of staging out and registering the data (if requested), submitting logging and accounting information, and eventually cleaning up the space used by the job. Client tools can also be used to retrieve job outputs at any time.

The reliable performance of such a system is achieved by using several innovative approaches:

- Usage of the specialized GridFTP server (see Section 3.1) for job submission provides for job and resource management flexibility, full GSI benefits, firewall-friendliness, and a lightweight client.
- The Grid Manager carries most of the workload, thus enabling a perfect front-end Grid model, where no Grid-related software (other than the LRMS) has to be installed on cluster worker nodes. The Grid Manager is highly configurable.
- The data management functionality of the Grid Manager includes reliable file transfer, input data caching and registration of output data to data indexing services via built-in interfaces.
- The system has no central element or service: the information tree is multirooted, and matchmaking is performed by individual clients.
- The information system appropriately describes the natural Grid entities and resources in a dedicated LDAP schema. The information about Grid jobs, users and resource characteristics is kept in local LDAP databases at every resource and is not cached in higher level databases: the imperative is to always have fresh data, or not to have it at all.
- There is no intermediate service between the client and the resource: the

Brokering Client performs the matchmaking and actual Grid job submission, as well as providing clients for other Grid services. Users can share the clients, or have several user interfaces per user.
- All the services are stateful, allowing easy failure recovery and synchronisation, when necessary.

Figure 1 shows several optional components, which are not required for an initial Grid setup, but are essential for providing proper Grid services. Most important are the *Storage Services* and the *Data Indexing Services*. As long as the users' jobs do not manipulate large amonts of data, these are not necessary. However, when Terabytes of data are being processed, they have to be reliably stored and properly indexed, allowing further usage. Data are stored at *Storage Elements* (SE), which in ARC come in two kinds: the "regular" disk-based SE is simply a GridFTP interface to a file system (see Section 3.4), while the Smart Storage Element (SSE) is a Web service providing extended functionality, including file movement and automatic file indexing (see Section 3.5). File indexing itself needs a specially structured database with a proper GSI-enabled interface. ARC currently has an interface to several data indexing systems: Globus Replica Catalog (RC) [16], Globus Replica Location Service (RLS) [17] and gLite Fireman [18]. RC and RLS were developed by the Globus project with the goal to facilitate cataloging and consequent locating of data sources. The original Replica Catalog was not GSI-enabled; however, ARC developers have added possibility to perform securely authenticated connections. Fireman is being development as a component of the gLite [19] Data Management System.

Another non-critical component that nevertheless often comes among the top-ranked in user requirements, is the *Grid Monitor* (Section 4.4). In the ARC solution, it is only a client without capabilities to manipulate resources or jobs, providing information only to the end-users. It relies entirely on the Information System, being basically a specialized user-friendly LDAP browser. Monitoring in ARC does not require special sensors, agents or dedicated databases: it simply re-uses the Information System infrastructure.

While monitoring offers real-time snapshots of the system, historical information has to be persistified for further analysis, accounting, auditing and similar purposes. To provide for this, a job's usage record is submitted by the Grid Manager to a *Logging Service* (Logger) (see Section 6.2). Each group of users that desires to maintain such records can run a dedicated Logger.

For all the structure described above to serve a particular user, this user should be known (authenticated) to the Grid and be authorised to use one or more services according to the policies adopted by the service providers. The ARC user management scheme is identical to most other Grid solutions: typically, Grid users are grouped in *Virtual Organisations* (VO), which provide their member lists via various *User Databases* or through a Virtual Organization Membership Service (VOMS [20]). ARC services that require authorisation, such as the Computing Service or a Storage Service, subscribe to a subset of these databases (depending on local policies) and regularly retrieve the lists of authorised user credentials.

The overall architecture does not impose any special requirements upon computing resources, apart from the preferable presence of a shared filesystem (*e.g.* NFS) between the head node and the compute (worker) nodes of a cluster. If there is no shared area, LRMS has to provide means for staging data between the head node and the compute nodes. Cluster-wise, ARC implements a pure front-end model: nothing has to be installed on the worker nodes.

Being developed mainly for the purposes of serial batch jobs, ARC presently does not support a range of other computational tasks. Most notably, there is no solution for an interactive analysis, such as graphical visualisation or other kind of tasks requiring real-time user input. Although it attempts to look to an end-user not much different from a conventional batch system, ARC has a substantial difference: the results of a batch job do not appear at the user's local area, until the user fetches them manually. The ARC job submission client does not provide an automatic job resubmission in the case of trivial failures (such as network outages), neither does it implement job re-scheduling from one computing service to another. Higher level tools or Grid services can be developed to assist in such tasks (e.g., the *Job Manager*, see Section 6). ARC is not suitable for jobs that are expected to execute instantly: this would require distributed advance resource reservation, which is not implemented in the Grid world. The related shortcoming is that ARC provides no solution for cross-cluster parallel jobs. All these limitations do not stem from the architectural solution: rather, they would require creating new dedicated services, quite possibly in the framework of the current design.

3 Services

ARC provides a reliable, lightweight, production quality and scalable implementation of fundamental Grid services such as Grid job submission and management, Grid resource management, resource characterization, resource aggregation, and data management.

A Grid enabled computing resource runs a non-intrusive Grid layer composed of three main services: specialized GridFTP (Sections 3.1), Grid Manager (Section 3.3), and the Local Information Service (Section 3.6.1). The first two constitute the Computing Service as such.

A Grid enabled storage resource makes use of either of the two types of Grid storage layers provided by ARC: a conventional solution, based upon the GridFTP server (Section 3.4), or the Web service based (Section 3.2) Smart Storage Element (Section 3.5). Storage resources also run Local Information Services.

Computing and storage resources are connected to the Grid via the registration service (Section 3.6.3) which links a resource to an Information Index Service (Section 3.6.2), this way implementing resource aggregation.

3.1 ARC GridFTP server

GridFTP interface was chosen. ARC provides its own GridFTP server (GFS), with a GridFTP interface implemented using the *Globus GSIFTP Control Connection* application programming interface (API) of the Globus Toolkit® libraries. The reason for introducing this server was an absence of GridFTP servers with flexible authorization. Only available implementation was modified WU-FTPD FTP server distributed as part of Globus Toolkit®With whole configurutation based on UNIX identity of users, it was incapable of providing adequate authorization in distributed environement.

GFS consists of two parts. The front-end part implements the GridFTP protocol, authentication, authorization and configuration of the service. The back-end part implements the file system visible through the GridFTP interface. This solution also adds possibility to serve various kind of information through GridFTP interface, and to reduce number of transport protocols used in system. And hence, to decrease complexity of software.

GFS does not implement all the latest features of the GridFTP protocols, but it is compatible with the tools provided by the Globus Toolkit® and other Grid projects based on it.

The front-end part takes care of identifying the connecting client and applying access rules (see Section 3.7). In the configuration, a virtual tree of directories is defined, with directories assigned to various back-ends. The activation of the virtual directories is performed in the configuration, according to the Grid identity of a client. Therefore, each client may be served by various combinations of services.

Information about Grid identity of user is propagated through whole chain down to back-ends. This make it possible to write back-end capable of making internal authorization decisions based entirely on Grid identity of connection client and avoid necessity to manage dedicated local user accounts.

The back-end part is implemented as a set of plugins. Each plugin is a dynamic library which implements a particular C++ class with methods corresponding to FTP operations needed to create a view of a virtual file system-like structure visible through the GridFTP interface.

Currently, the ARC distribution comes with three plugins. Two of them, called *fileplugin* and *gaclplugin*, implement different ways for accessing local file systems (see Section 3.4 for description). Their virtual file system almost corresponds to the underlying physical one. The third plugin is called *jobplugin*: it generates a virtual view of session directories managed by the Grid Manager service and is described in Section 3.3.

3.2 HTTPSd framework

ARC comes with a framework for building lightweight HTTP, HTTP over SSL (HTTPS) and GSI (HTTPg) services and clients.[3] It consists of C++ classes and a server implementation. These classes implement a subset of the HTTP protocol, including GET, PUT and POST methods, content ranges, and other most often used features. The implementation is not feature rich, instead, it is light-weight and targeted to transfer the content with little overhead. It has some support for gSOAP [22] integrated in the implementation of the POST method, thus allowing creation and communication with Web Services. This framework does not support message-level security.

The server implements configuration, authentication and authorization in a way common for all the network services of ARC. It listens to two TCP/IP ports for HTTPS and HTTPg connections. The configuration assigns various plugin services to paths. The services themselves are dynamic libraries which provide configuration and creation functions and supply an implementation of a C++ class responsible for handling HTTP requests.

The client part is a thin layer which allows one to pump data through GET and PUT methods, and provides functionality adequate to the server part. The DataMove library, described in Section 5.2, uses its functionality to provide a multi-stream access to HTTP(S/g) content.

ARC is currently being delivered with two HTTP(S/g) plugins. Those are two Web Services: the *Smart Storage Element* (Section 3.5) and the *Logger* (Section 6.2). Some others are being developed. Among them is a Web Service for job submission and control, intended to replace the *jobplugin* of the GFS. It also provides an interface for ordinary Web browsers to monitor the execution of Grid jobs.

[3] For a summary of these mechanisms, see [21].

The purpose of the ARC Grid Manager (GM) is to take necessary actions on a computing cluster front-end, needed to execute a job. In ARC, a job is defined as a set of input files, a main executable, the requested resources (including pre-installed software packages) and a set of produced files. The process of gathering the input files, executing the main executable, and transferring/storing the output files is carried out by GM (Figure 2).

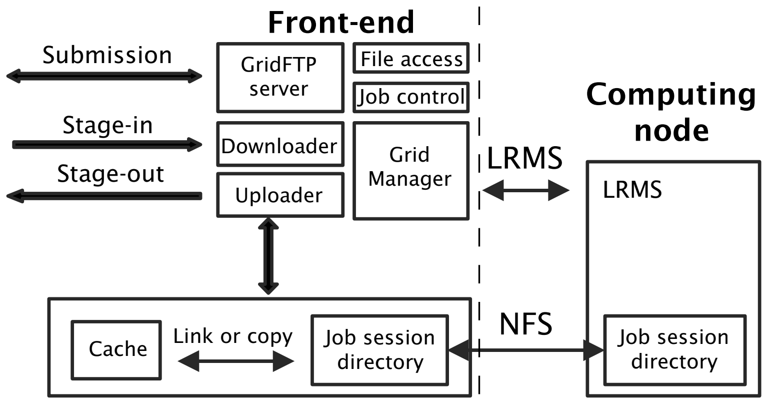

Fig. 2. A typical setup of ARC Grid Manager

GM processes job descriptions written in Extended Resource Specification Language (XRSL, see Section 4.1).

Each job gets a directory on the execution cluster, the *session directory*. This directory is created by GM as a part of the job preparation process. The job is supposed to use this directory for all input and output data, and possibly other auxilliary files. All the file paths in the job description are specified relative to the session directory. There is no notion of the user's home directory, nor user ID, nor other attributes common in shell access environments. Jobs executed on ARC enabled clusters should not rely on anything specific to user account, not even on the existence of such an account.

In most common installations, the space for the session directory resides in an area shared by means of the Network File System with all the computing nodes. In this case, all the data that a job keeps in the session directory is continuously available to the user through the *jobplugin* of the GFS, as described below.

It is also possible to have a session directory prepared on the front-end computer and then moved to a computing node. This limits the amount of remotely available information during the job execution to that generated by GM itself.

Input files are gathered in the session directory by the *downloader* module of GM. In the most common configuration this module is run by the GM directly on the front-end. To limit the load of the front-end machine, it is possible to control the number of the simultaneous instances of the downloader modules and the number of downloadable files. GM can be instructed to run downloader on a computing node as a part of the job execution. Such a configuration is desirable on computing clusters with slow or small disk space on the front-end.

GM supports various kinds of sources for input as well as for output files. These include pseudo-URLs expressed in terms of data registered in Indexing Services. All types of supported URLs can be found in Section 5.2.

A client can push files to the session directory as a part of the job preparation. This way, the user can run remotely a snapshot of an environment prepared and tested on a local workstation without the trouble of arranging space on storage services.

GM handles a cache of input files. Every URL requested for download is first checked against the contents of the cache. If a file associated with a given URL is already cached, the remote storage is contacted to check the file's validity and the access permissions based on the user's credentials. If a file stored in cache is found to be valid, it is either copied to session directory, or a soft-link is used, depending on the configuration of GM. The caching feature can be turned off either by the resource administrator or by the user submitting the job.

GM can automatically register the files stored in its cache to a Data Indexing Service. This information is then used by a brokering client (see Section 4.2) to direct jobs into a cluster with a high probability of fast access to input data. This functionality has proved to be very useful when running many jobs which process the same input data.

It is possible to instruct GM to modify the files' URLs and hence access them through more suitable protocols and servers. This feature makes it possible to have so called "local Storage Elements" in a way transparent to users. Together with the possibility to have this information published in the Information System (see Section 3.6), this may significantly reduce the job preparation costs.

After the contents of the session directory has been prepared, GM communicates with the Local Resource Management System (LRMS) to run the main executable of the job. GM uses a set of back-end shell scripts in order to perform such LRMS operations as job submission, detection of the exit state of the main executable, etc. Currently supported LRMS types include various PBS [23] flavors, Condor [1] and SGE [24]. There is also support for job execution on the front-end, meant for testing purposes only.

The output files generated by the job have to be listed in the job description together with their final destinations, or indicated to be kept on the computing resource for eventual retrieval. After the job's main executable has stopped running, they are delivered to the specified destinations by the *uploader* module. The uploader is similar to the downloader described above.

The session directories are preserved only temporarily after the session is finished. Hence, the user can not directly re-use any produced data in sequential jobs and has to take care of retrieving output files kept on the computing resource as soon as possible.

Support for software *runtime environments* (RTE, see also Section 4.1) is also a part of GM. RTEs are meant to provide a unified access to pre-installed software packages. In the GM, support for RTE is made through shell scripts executed a few times during job duration: after the preparation phase of the job, before running the main executable, and after it has finished. The scripts may set up environments needed by pre-installed software packages, and even modify the job description itself. To standardize creation of new RTEs and to make it easier for users to find out how to use available RTEs, a registration portal was established [25] by the NorduGrid partner, the Nordic Data Grid Facility.

A job passes through several states in GM, and each state can be assigned an external plugin. This way, the functionality of GM can be significantly extended with external components to include e.g. logging, fine-grained authorization, accounting, filtering, etc.

GM itself does not have a network port. Instead, it reads the information about job requests from the local file system. This information is delivered through the GridFTP channel by using a customized plugin for the ARC GridFTP server, named *jobplugin*. Information about the jobs served by GM and content of their session directories is represented by the same plugin through the virtual file system. Session directories are mapped to directories, and actions to various FTP commands and files. Access to every piece of information is protected, and a flexible access control over the job data is possible. By default, every piece of information about the job is accessible only by a client which presents the same credentials as those used to submit a job. By adding GACL [26] rules to job description, a user can allow other users to see results of the job and even to manage it. Thus users can form groups to share responsibility for submitted jobs.

GM has several other, less significant, features. Those include E-mail notification of job status changes, support for site local credentials through customizable plugins, reporting jobs to centralized logging service (see Section 6.2), etc.

Most Storage Elements on sites equipped with ARC are GridFTP servers. A significant number of them use ARC GridFTP server with one of the following plugins.

The *fileplugin* presents the underlying file system in a way similar to ordinary FTP servers with file access based on a mapping between the client certificate subject and a local UNIX[4] identity. It is also possible to specify *static* access restrictions for pre-defined directories directly in the configuration of the GridFTP server (GFS). In addition, there is a way to completely disregard UNIX identities of the clients, thus relying only on capability of GFS to create virtual file system trees, or to mix both ways.

This plugin is meant to be used as a replacement for an ordinary FTP server in user mapping mode, or for user communities which do not need complex dynamic access control inside dedicated storage pools.

The *gaclplugin* uses GACL [26], an XML-based access control language, in order to control access to files and directories on a storage element. Each directory and file has an associated GACL document stored in a dedicated file. In order not to break the compatibility with the FTP protocol, modification of GACL rules is performed by downloading and uploading those files. GACL files themselves are not visible through FTP file listing commands.

Each GACL document consists of entries with list of identities and access rules associated with them. There is no predefined set of supported types of identities. Currently, ARC can recognize a X.509 certificate subject (Distinguished Name), VOMS [20] names, groups, roles and capabilities, and membership of a predefined Virtual Organization (lists of members must be created by a third-party utility). GACL permissions control possibility to `read` and `write` objects, `list` the contents of directories or get information about stored file. The `admin` permission allows one to obtain and modify the contents of GACL documents.

A GridFTP server with the *gaclplugin* is most convenient for providing flexible dynamic access control based purely on Grid identities.

[4] UNIX is a trademark of the Open Group.

A “Smart” Storage Element (SSE) is a data storage service that performs a basic set of the most important data management functions without user intervention. It is not meant to be used in an isolated way. Instead, its main purpose is to form a data storage infrastructure together with similar SSEs and Data Indexing Services, such as the Globus® Replica Catalog or Replica Location Service.

The main features of SSE and the data storage infrastructure which can be implemented using SSE are as follows (see also Figure 3):

- SSE can be used as a stand-alone service, but preferably should be a part of a full infrastructure and must be complemented with Data Indexing Services (DIS).
- DIS can be either a common purpose or an application specific one. Modular internal architecture of SSE allows adding (at the source code level) interfaces to previously unsupported DIS types.
- SSE has no internal structure for storing data units (files). Files are identified by their identity used in a DIS (Logical File Name, GUID, Logical Path, etc.)
- All operations on data units (creation, replication, deletion of replica, etc.) are done upon a single request from a client through SSE. Hence most operations are atomic from the client’s point of view. Nevertheless, objects created during those operations do not always have final state and may evolve with time. For example, a created file can be filled with data, then validated, and then registered.
- Entry points to a DIS infrastructure should be used by clients only for operations not involving data storage units. If an operation involves any modification of SSE content which has to be registered in DIS, any communication to DIS will be done by the SSE itself.
- Access to data units in SSE is based on a Grid Identity of the client and is controlled by GACL rules, for flexibility.
- Data transfer between SSEs is done upon request from a client, by SSEs themselves (third-party transfers using “pull” model).

An interface to SSE is based on SOAP over HTTPS/HTTPg (Web Services) with data transfer through pure HTTPS/HTTPg. It is implemented as a plugin for the HTTPSd framework (see Section 3.2). SOAP methods implemented in SSE allow it to:

- Create a new data unit and prepare it to be either filled by the client, or its contents to be pulled from an external source. The SSE can pull data both from inside of the infrastructure (*replication*) and from external sources. SSE

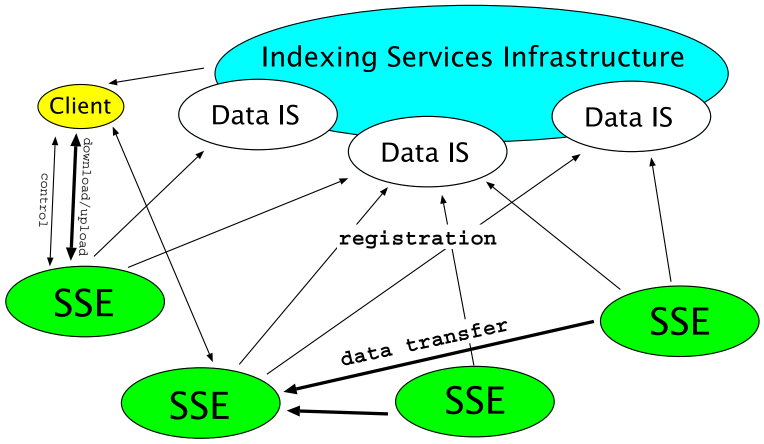

Fig. 3. An example of an architecture of a Data Storage Infrastructure.

uses the DataMove library (see Section 5.2) for data transfer and supports all the corresponding protocols.
- Get information about stored data units — *regular expression* search for names is possible.
- Inspect and modify the access rules of data units.
- Update the meta data of stored units. This method is mostly used to provide checksums needed for content validation.
- Remove data units.

Recently, a SRM v1.1 [27] interface has been implemented for SSE. Work to provide support for the current standard SRM v2.1.1 is in progress.

3.6 Information system

The ARC middleware implements a scalable, production quality, dynamic, LDAP-based distributed information system via a set of coupled resource lists (Index Services) and local LDAP databases. Implementation of the ARC information system is based on OpenLDAP ([28], [29], [30]) and its modifications provided by the Globus® MDS framework [31,32]. NorduGrid plans to replace the Globus® LDAP modifications with native OpenLDAP functionalities.

The system consists of three main components:

(1) Local Information Services (LIS),
(2) Index Services (IS),
(3) and Registration Processes (RP)

The components and their connections shown in the overview Figure 4 are described in the following subsections.

3.6.1 Local Information Service

Local Information Services (LIS) are responsible for resource (computing or storage) description and characterization. We note that the description of Grid jobs running on a resource is also a part of the resource characterization: job status monitoring within ARC is done via querying the LIS components of the information system. The local information is generated on the resource, and optionally is cached locally within the LDAP server. Upon client requests, the information is presented to the clients via LDAP interface. A LIS is basically nothing more but a specially populated and customized OpenLDAP database. Figure 5 together with the description below gives a detailed account of the internal structure of the LIS.

The dynamic resource state information is generated on the resource. Small and efficient programs, called information providers, are used to collect local state information from the batch system, from the local Grid layer (e.g., the Grid Manager or the GridFTP server), or from the local operating system (e.g., information available in the `/proc` area). Currently, ARC infoproviders are interfaced with the same systems as GM, namely the UNIX fork, the PBS family [23] (OpenPBS, PBS-Pro, Torque), Condor [1] and Sun Grid Engine (SGE) [24] batch systems. The output of the information providers (generated in LDIF format) is used to populate the LIS. A special OpenLDAP back-end, the Globus® GRIS [32], is used to store the LDIF output of the information providers. The custom GRIS back-end implements two functionalities: it is

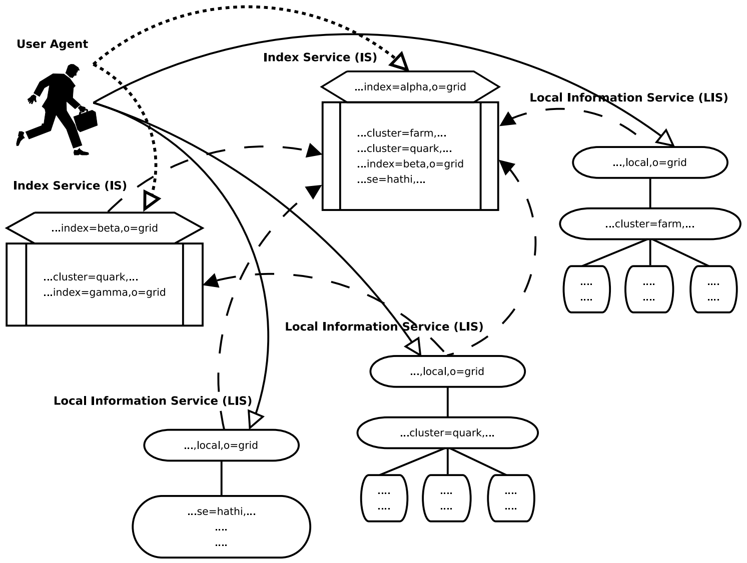

Fig. 4. Overview of the ARC information system components. Clients issue type-1 queries (dotted arrows) against Index Services which maintain a dynamic list of contact URLs of LISs and further Index Services registering to them (registration processes are marked by dashed arrows). Then clients perform type-2 queries (solid arrows) to discover resource properties stored in the LISs.

capable of caching the providers' output, and upon client query it triggers the information providers unless the data is already available in the cache. The caching feature of the OpenLDAP back-end provides protection against overloading the local resource by continuously triggering the information providers.

The information stored in the LISs follows the ARC information model. This model gives a natural representation of Grid resources and Grid entities. It is implemented via an LDAP schema by specifying an LDAP Directory Information Tree (DIT). The schema naturally describes Grid enabled clusters, queues and storage elements. The schema, unlike other contemporary Grid schemas, also describes Grid jobs and authorized Grid users. A detailed description of this information model is given in the online manual [33].

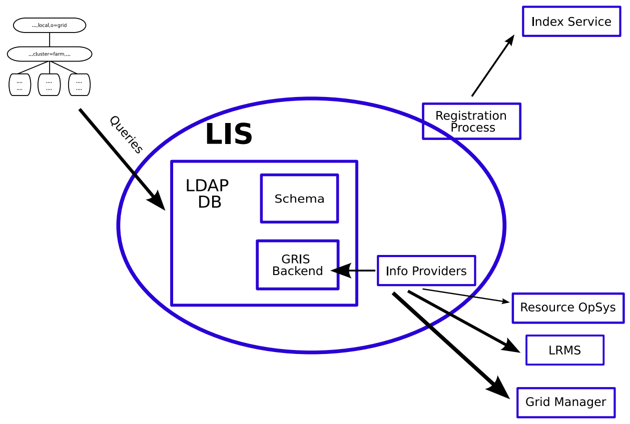

Fig. 5. Internal structure of the Local Information Service

3.6.2 Index Services

Index Services are used to maintain dynamic lists of available resources, containing the LDAP contact information of the registrants. A registrant can either be a LIS or another Index Service. The content of the Index Service, that is, the information about the registrants, is pushed by the registrants via the registration process and is periodically purged by the Index Service, this way maintaining a dynamic registrant list. Index Services are implemented as special purpose LDAP databases: the registration entries are stored as pseudo-LDAP entries in the Globus GIIS LDAP back-end [32]. Although the registration information is presented in a valid LDIF format, the registration entries do not constitute a legitimate LDAP tree, therefore they are referred to as the pseudo-LDAP registration entries of the Index Service. A specially crafted LDAP query is required to obtain the registration entries. Besides the registrant tables, no any other information is stored in the ARC indices. The current implementation based on such misuse of the GIIS-backend is planned to be replaced by something more suitable: the simplistic functionality of maintaining a dynamic list of registrant tables could be trivially done e.g. in native LDAP. Furthermore, ARC Index Service does not implement any internal query machinery, Index Services are basically plain lists of contact URLs of other Index Services and LISs.

3.6.3 Registration Processes, Topology

Individual LISs and Index Services need to be connected and organized into some sort of a topological structure in order to create a coherent information system. ARC utilizes registration processes and Index Services to build a distributed information system of the individual LISs and Index Services.

During the registration process, the registrant (lower level) sends its registration packet to an Index Service. The packet contains information about the host (the registrant) initiating the process and the type of the information service running on the registrant (whether it is a LIS or an IS). The registration message is basically an LDAP contact URL of the information service running on the registrant. Registrations are always initiated by the registrants (bottom-up model) and are performed periodically, thus the registration mechanism follows a periodic push model. Technically, the registrations are implemented as periodic `ldapadd` operations.

The LISs and the Index Services of ARC are organized into a multi-level tree hierarchy via the registration process. The LISs that describe storage or computing resources represent the lowest level of the tree-like topology. Resources register to first level Index Services, which in turn register to Second level Services, and so forth. The registration chain ends at the Top Level Indices which represent the root of the hierarchy tree. The structure is built from bottom to top: the lower level always registers to the higher one. The tree-like hierarchical structure is motivated by the natural geographical organization, where resources belonging to the same region register under a region index, region indices are registering to the appropriate country index, while country indices are grouped together and register to the top level Grid Index Services. Besides the geographical structuring there are some Index Services which group resources by specific application area or organization. These application/organization Index Services either link themselves under a country Index or register directly to a Top Level Index. In order to avoid any single point of failure, ARC suggests a multi-rooted tree with several top-level Indices. Figure 6 shows a simplified schematic view of the hierarchical multi-rooted tree topology of ARC-connected resources and Index Services.

3.6.4 Operational overview

Grid clients such as monitors, Web portals or user interfaces (Section 4) perform two types of queries against the Information System:

(1) During the *resource discovery process* clients query Index Services in order to collect list of LDAP contact URLs of Local Information Services describing Grid resources.
(2) During a *direct resource query* the clients directly contact the LISs by making use of the obtained LDAP contact URLs.

Once the client has collected the list of resource LDAP contact URLs (by performing type-1 queries against the Index Services), the second phase of the information collection begins: type-2 queries are issued against LISs to gather resource descriptions. Both types of queries are carried out and served via the LDAP protocol. Unlike other similar purpose systems (Globus Toolkit® 2 GIIS or Globus Toolkit® 4 aggregator [34], or R-GMA [35]), ARC has no service which caches or aggregates resource specific information on a higher level. ARC Index Services are not used to store resource information, they only maintain dynamic list of LDAP URLs. Figure 4 presents an overview of the ARC information system components showing relations between the Index Services, LISs registration processes and the two type of queries.

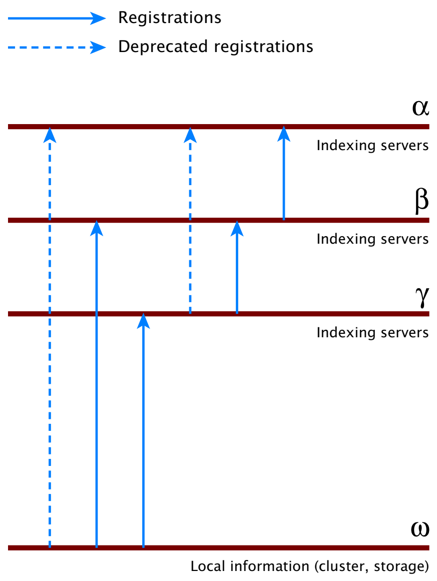

Fig. 6. Resources (LIS) and Index Services are linked via the registration process creating a hierarchical multi-rooted tree topology. The hierarchy has several levels: the ω lowest level (populated by LIS), the γ organisation/project level, the β country/international project level and the α top (root) Grid level. The tree is multi-rooted, such that there are several α-level Index Services and in bigger countries there are several β- or country-level Index Services.

The security model of the ARC middleware is built upon the Grid Security Infrastructure (GSI) [3]. Currently, for communication ARC services do not use message level security, but rather transport level security (see [21]).

The ARC services that make use of authentication and authorization are those built on top of the GridFTP server and the HTTPSd framework.

The authentication part makes use of Certificate Authorities (CA) signing policies, whereby sites can limit the signing namespace of a CA. Moreover, ARC provides a utility (`grid-update-crls`) to retrieve certificate revocation lists on a regular basis.

In a basic and largely obsolete authorization model, ARC services support a simple X.509 certificate subject (Distinguished Name) mapping to UNIX users through the *grid-mapfile*. Automatic *grid-mapfile* creation can be done using the `nordugridmap` utility which can fetch Virtual Organization membership information from LDAP databases, and simple X.509 subject lists — through FTP and HTTP(s). In addition, it is possible to authorize local users, specify filters and black-lists with this utility.

For advanced authorization, most of the ARC services are capable of acting purely on Grid identity of the connecting client without relying on local identities. Those services support a unified set of rules which allows to identify users, group them and choose adequate service level.

In addition to personal Grid identity of a user, other identification mechanisms are supported. Among them are the Virtual Organization Membership Service (VOMS) and general VO membership (through external information gathering tools). It is also possible to use external identification plugins, thus providing high level of flexibility.

In addition to access level authorization, many services make use of the client Grid identities internally to provide fine-grained access control. Currently, all such services use GACL language for that purpose. Fine-grained access control is available in the GridFTP server's file access and job control plugins and in the SSE service.

The job handling service — the Grid Manager — also has a pluggable authorization mechanism that can provide fine-grained authorization and accounting during job execution (see Section 3.3) through third-party plugins.

4 User interfaces and clients

Users interact with the ARC services on the Grid resources using various client tools. With the help of the basic command line interface (CLI, Section 4.2), users can submit jobs to the Grid, monitor their progress and perform basic job and file manipulations. Jobs submitted through the command line interface must be described in terms of the Extended Resource Specification Language (Section 4.1).

The basic command line interface for job handling is similar to a set of commands that can be found in a local resource management system. In addition to this, there are commands that can be used for basic data management, like creating and registering files on storage elements and administering access rights to the data.

In addition to the CLI, the ARC client package includes some commands from the Globus Toolkit®, in particular, those needed for certificate handling and data indexing (see Section 4.3).

Several other clients interacting with the ARC services are developed as well. An example of such is the ARC Grid Monitor (see Section 4.4) that presents the information from the Information System using a Web browser.

4.1 Resource Specification Language

The XRSL language used by ARC to describe job options is an extension of the core Globus RSL 1.0 [36]. The extensions were prompted by the specifics of the architectural solution, and resulted in the introduction of new attributes. In addition, differentiation between the two levels of job options specifications was introduced. The *user-side XRSL* is a set of attributes specified by a user in a job description file. This file is interpreted and pre-processed by the brokering client (see Section 4.2), and after the necessary modifications is passed to the Grid Manager (GM). The *GM-side XRSL* is a set of attributes prepared by the client, and ready to be interpreted by the GM.

The purpose of XRSL in ARC is therefore dual: it is used not only by users to describe job requirements, but also by the client and GM as a communication language.

The major challenge for many applications is pre- and post-staging of considerable amount of files, often of a large size. To reflect this, two new attributes were introduced in XRSL: *inputFiles* and *outputFiles.* Each of them is a list of local-remote file name or URL pairs. Local to the submission node input files are uploaded to the execution node by the client; the rest is handled by the GM. The output files are moved upon the job completion by the GM to a specified location (Storage Element). If no output location is specified, the files are expected to be retrieved by a user via the client.

Another important addition is the *runtimeenvironment* attribute. Very often jobs require specific local environment in order to get properly executed. This includes not merely — and not necessarily — a presence of a specific pre-installed software, but also proper settings of local variables and paths, existence of certain services or databases, a particular system configuration and so on. It is convenient to denote such a suite of settings with a single tag — a runtime environment name. This tag is validated and advertised by the resource owners and simplifies matchmaking, since users can request the necessary runtime environment tag by the means of XRSL.

Several other attributes were added in XRSL, allowing users to specify in details their preferences [37]. A typical XRSL file is shown below:

```
& (executable="ds2000.sh") (arguments="1101")
(stdout="dc1.002000.simul.01101.hlt.pythia_jet_17.log") (join="yes")
(inputfiles=
   ("ds2000.sh"
"http://www.nordugrid.org/applications/dc1/2000/dc1.002000.simul.NG.sh"))
(outputFiles=
   ("dc1.002000.simul.01101.hlt.pythia_jet_17.log"
"rc://dc1.uio.no/2000/log/dc1.002000.simul.01101.hlt.pythia_jet_17.log")
   ("atlas.01101.zebra"
"rc://dc1.uio.no/2000/zebra/dc1.002000.simul.01101.hlt.pythia_jet_17.zebra")
   ("atlas.01101.his"
"rc://dc1.uio.no/2000/his/dc1.002000.simul.01101.hlt.pythia_jet_17.his")
   ("dc1.002000.simul.01101.hlt.pythia_jet_17.AMI"
"rc://dc1.uio.no/2000/ami/dc1.002000.simul.01101.hlt.pythia_jet_17.AMI")
   ("dc1.002000.simul.01101.hlt.pythia_jet_17.MAG"
"rc://dc1.uio.no/2000/mag/dc1.002000.simul.01101.hlt.pythia_jet_17.MAG")
)
(jobname="dc1.002000.simul.01101.hlt.pythia_jet_17")
(runTimeEnvironment="DC1-ATLAS")
```

4.2 ARC Command Line Interface

The ARC command line interface (CLI) provides basic tools for simple Grid job management and single file manipulation. It also incorporates Grid resource discovery, matchmaking, and some brokering functionality. The ARC CLI provides all the tools necessary to work in the Grid environment and yet remains small in size and simple in installation. A user can run several CLIs: clients installed on different workstations can be used simultaneously and interchangeably by the same user without the necessity to instantiate them as permanently active services.

4.2.1 Job management

A user should be able to submit jobs to Grid without having to worry about which particular cluster will execute a job. From a user's perspective it should not be much different from submitting a job to a local cluster. The set of commands used for job management in the ARC CLI is therefore similar to a set of commands provided by a local resource management system on a cluster. The commands are listed in Table 1.

Table 1
The job management commands in the ARC user interface.

ngsub	job submission
ngstat	find information about resources and jobs
ngcat	catenate stdout, stderr or the grid manager's error log of a job
ngget	retrieve the results of a finished job
ngkill	kill a running job
ngclean	clean up after a job (deletes the session directory on the cluster)
ngrenew	renew the GSI proxy certificate for a job
ngresub	resubmit the job (using the same XRSL) to a different cluster
ngsync	synchronise the local job data base using the Information System
ngtest	perform various tests

Job submission is done using the `ngsub` command which invokes a built-in brokering mechanism. Jobs that are submitted must be described using the XRSL specification language (Section 4.1). When a job description is given to `ngsub` its requirements are compared to the capabilities of the computing resources that can be discovered through the Information System. A brokering between the requirements and the capabilities is performed and a queue that satisfies the given criteria is selected as submission target. The `ngsub` client then modifies the *user-side XRSL* by resolving optional arguments before sending the job description to the selected target as a *GM-side XRSL*.

The brokering capabilities in the ARC middleware are distributed since each client acts as its user's individual broker, thereby avoiding the possible single point of failure and potential bottleneck that would exist if a centralised broker was used.

Several attributes in the XRSL are used in the brokering. A specific cluster or queue can be selected by providing the corresponding XRSL attributes. The requested number of CPUs, memory, architecture, operating system or middleware version and pre-installed runtime environments are compared to the capabilities published by the cluster in the information system and those targets that fail to meet the specified requirements are rejected.

The brokering algorithm takes into account the size of the input files and where they are located. It tries to minimise the amount of data that has to be transferred from storage elements that are not local to the submission target. It resolves the location of files registered in file index servers like RC and RLS and if cache registration is enabled at the clusters, the information about cached files is used as well. If there are no free CPUs available on a cluster where the data is already present, the job description is sent to a different cluster, thereby triggering the transfer of the data to that cluster.

The `ngstat` command is a user tool that queries the Information System and presents the result to the user. It can be used for both getting information about available computing resources and their capabilities and about the status of jobs submitted to the system.

There are also commands for killing queueing or running jobs, for retrieving outputs of finished jobs and for other job management tasks.

4.2.2 Data Management

Most of the Data Management tasks are automatic from a user's perspective since the file movement and registration is performed on the user's behalf by the Grid Manager, provided the correct attributes are specified in the XRSL job description.

There are also user tools that interact with the Storage Services and Data Indexing Services directly (see Table 2). These commands also provide for a higher level data management since they support pseudo-URLs [38] referring to Index Server registrations. If such pseudo-URLs are used, file registration in the Data Indexing Service will be modified in a consistent way to reflect the changes in the Storage Services.

Table 2
Data management commands in the ARC CLI

ngcopy	copy files to from or between Storage Servers
ngremove	delete files from Storage Servers
ngls	list files on Storage or Index Servers
ngacl	set or get the access control list of a file

4.3 External commands

The ARC middleware is built on top of the pre-WS components of the Globus Toolkit®. The toolkit provides command line interfaces to some of the components used along with ARC.

In particular, the ARC middleware uses the Globus Security Infrastructure (GSI) for authentication. Users therefore make use of the toolkit's command line tools that handle creation and management of proxy certificates, such as `grid-proxy-init`.

Several Globus® data management technologies have been integrated into the ARC middleware, such as the Replica Catalog and the Replica Location Service. In order to perform data management, users should use the clients of these services. Most notably, the `globus-rls-cli` tool is very useful for manipulating and querying registered file attributes in the Replica Location Service.

4.4 The Monitor

A reliable and up-to-date information about the Grid system, its status, its components, users and even single tasks is provided by the means of the ARC Grid Monitor [39,40]. The Monitor is an end-user tool, offering a simple way of checking the status of the Grid.

The ARC Grid Monitor is a Web interface to the ARC Information System (Section 3.6), allowing to browse all the data published through it. The Information System provides a robust and dynamic model for accessing not only quasi-static information about resources and services, but also about such rapidly changing parameters like queue and job status. Being based on OpenLDAP, it can be easily interfaced to any browsing or monitoring tool, giving thus a user-friendly overview of all the resources. The Grid Monitor makes use of the PHP4[5] LDAP module to provide a real-time monitoring and initial debugging via any Web browser.

The structure of the Grid Monitor to great extent follows that of the Information System. The basic objects are defined by the schema's objectclasses. For each objectclass, either an essential subset of attributes, or the whole list of them, is presented in an easily accessible inter-linked manner. This is realized as a set of browser windows, each being associated with a corresponding module.

[5] For details about PHP, Hypertext Preprocessor language see e.g. [41].

Each module displays both dynamic and static information: for example, a queue name is static, while the amount of running jobs in this queue is dynamic. Most of the displayed objects are linked to appropriate modules, such that with a simple mouse click, a user can launch another module, expanding the information about the corresponding object or attribute. Each such module opens in an own window, and gives access to other modules in turn, providing thus a very intuitive browsing.

The Grid Monitor's framework allows full localization, making use of the browsers' preferred language settings. Currently, the Monitor is available in 5 languages, being perhaps the only multilingual Grid tool.

The Monitor layout is deliberately simplistic, ensuring its smooth working with any Web browser. It has been used via a variety of such, starting from text-based Lynx and ending with a mobile phone browsers. This effectively allows ARC users to interact with the Grid literally by phone.

5 Developer interfaces

The public interfaces of ARC-enabled resources are expressed via standard open protocols: GridFTP, LDAP and HTTP(S). Any client capable of talking LDAP and GridFTP can interface to all ARC computing and most storage resources. In order to ease this communication and to hide the low-level details of the LDAP schemas and GridFTP internals (e.g. job submission, control commands) ARC provides a set of developer libraries. The libraries also include convenient solutions for handling GSI identities and XRSL processing, among others. The next subsection describes the ARClib library, focusing on job submission including brokering, while Section 5.2 introduces a common purpose data transfer library.

5.1 ARCLib and JARCLib

ARCLib is a client library for ARC written in C++ with a well-defined API. The library consists of a set of classes for

- handling proxy, user and host certificates,
- discovering and querying resources like clusters and storage elements,
- handling XRSL job descriptions,
- operating on GridFTP servers — uploading and downloading files etc,
- brokering and
- job submission.

ARCLib also incorporates a set of common useful functionalities suitable for both clients and servers, like URL handling, conversion routines for dates and times and so on. ARCLib is complete enough to perform most client operations but is still very simple to use. Furthermore, ARCLib is wrapped using SWIG [42] and thus also provides a Python [43] interface to the API. ARCLib is therefore well suited for writing client utilities, graphical user interfaces, and portals.

ARCLib depends on pre-WS Globus Toolkit® for certificate handling, RSL descriptions and interacting with GridFTP servers, and on OpenLDAP for discovering and querying the information system for resources. Otherwise the library is self-contained.

Resource discovery and querying in ARCLib is done by standard LDAP queries to the information system, but the whole process of discovering and querying resources is wrapped in simple method calls. For example, getting information about all the clusters or all the storage elements registering to the ARC information system can be done by one simple method call. ARCLib does fully threaded LDAP queries to all resources in one go, thus providing maximum speed and performance.

Brokering is the process of selecting the most suitable resource in accordance with the job specification. Having obtained a list of candidate target resources, the brokering step consists of removing the candidates inconsistent with the user's job specification and sorting the remaining candidates after some criteria. ARCLib finds the candidate targets by a full resource discovery and querying the information system, and then removes inconsistent targets after criteria like requested memory on the nodes of the clusters, requested CPU time, requested installed runtime environments and so on. It then sorts the remaining targets after the speed of the CPU's of the clusters.

The ARCLib brokering framework makes it possible for the users to write their own brokers and include them in the brokering. At the moment, this requires re-compilation of a part of ARCLib but it is planned to make it possible to dynamically load user-provided brokers.

Job submission in ARC is done by uploading the corresponding XRSL job description to a specific directory on a cluster's GridFTP server. From here, the Grid Manager (GM) on the cluster takes over and handles the job. Job submission in ARCLib follows logically after the brokering step. Having obtained a sorted list of possible targets from this brokering step, the ARCLib methods try to submit the job description to the first target in the target list. If successful, they upload the corresponding local input files specified in the job description and get back the job ID assigned by the GM on the cluster. This job ID is returned to the user. If job submission to the first target fails, the second one is tried and so on.

JARClib is a thin layer over GridFTP and LDAP protocols written in Java language. Its aim is to provide a basic ARC API to Java programmers, in the same way as ARCLib provides an API for C/C++ programmers. This enables Java programmers to create platform independent ARC clients, including applets and Java standalone applications.

Currently, JARCLib supports job submission and control through the GridFTP interface, basic Information System interfacing methods (e.g. computing resource discovery) and basic matchmaking. All of these actions can be executed in parallel with a possibility to specify the maximum timeout for the whole operation. JARCLib's implementation relies on Globus® CoG toolkit [44] and Netscape Directory SDK [45].

5.2 DataMove

DataMove is a common purpose data transfer library used by most services and utilities of the ARC middleware. This way, the capabilities of all parts of ARC are unified. The DataMove library is implemented in C++. It does not provide a POSIX-like interface; instead it is meant to be used for creating data transfer channels to stream data through an application.

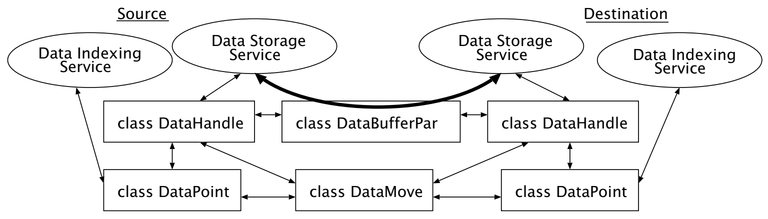

Fig. 7. Interaction between classes and external services for transfering data from "source" to "destination".

DataMove consists of an interface for URL abstraction (`DataPoint`) with integrated support for data registration in Data Indexing Service. Figure 7 illustrates typical usage for transfering data between two endpoints. Data access interface (`DataHandle`) has an expandable support for popular data transfer protocols. An abstraction of memory buffer (`DataBufferPar`) is used to establish a multi-stream channel between two `DataHandle` instances. Other features include getting information about objects, control over transfer speed, checksum calculation, etc.

Currently supported services are: Globus® RC and RLS, gLite Fireman [18], GridFTP, HTTP wrapped in SSL and GSI, SRMv1.1 [46], SSE and local file systems.

6 Ongoing development

As ARC is nearing its 0.6 release, new functionalities are being added. In the next paragraphs, the Job Manager and the HTTPS job submission framework development are described in brief. Then, the Java Graphical User Interface (can be used as an applet as well) is being introduced, followed by the Logger framework and the application portals.

The Job Manager is a higher-level client tool for managing jobs on ARC enabled Grids. It is able to supervise jobs by monitoring their states and reacting to changes in the state. The Job Manager is constructed in such a way that a user can tailor it to fit specific needs for controlling Grid jobs by plugging in handlers for different situations. The system acts as a layer between the user and the Grid, i.e., the users interact with Grid through the Job Manager.

As a first step to a Web service based job control interface, a new experimental service is being developed. It is going to provide a replacement for job control over GridFTP currently used in ARC middleware. The interface fully supports the authentication and authorization used in ARC and uses a lightweight (comparing to GridFTP) secure HTTP(S/g) channel. A pure Web interface to session directories and job information is also available.

6.1 Java GUI

The Java GUI client for the ARC middleware, the “Arconaut”, is currently in the development stage. It is built on top of JARCLib (see Section 5.1). The main aim of the design was to provide a localizable user friendly interface to credential management, job description generation, submission and management of the jobs.

The main features of the Arconaut are as follows:

- creating jobs for specific applications (such as Scilab, Povray or any other engine based applications) completely transparently with respect to the underlying XRSL job description;
- creating or importing a user proxy from a number of sources (including MyProxy servers [47]);
- submitting a job to Grid with a single click, or browse manually for the computing resource that suits the job the best;
- issuing standard actions on multiple submitted jobs in an intuitive way.

Arconaut can be used either as a Java Web Start application or as a standalone application. It is platform independent as it depends only on pure Java libraries. The modular architecture provides an easy way for extending the functionality of Arconaut.

6.2 Logging service

ARC provides a framework to deploy a Grid-wide logging infrastructure. The logging system consists of a Logging service (Logger) and logging clients.

The Logging Service (Logger) is a MySQL database with a Web service interface implemented in the HTTPSd infrastructure (Section 3.2). The Logger database collects Usage Records (UR) of Grid jobs. The URs are produced on the Grid-enabled computing resources and pushed to the Logger database in a SOAP message. Access to the Logger database is controlled via flexible common access rules which provide administrative and logging access levels. For ordinary clients, access to every UR is private.

The Usage Records stored in the Logger database describe Grid job characteristics including resource usage, execution information, ownership, etc. The current UR of ARC is very simplistic and contains only very basic information about jobs. A new UR based on the corresponding Global Grid Forum (GGF) [48] documents has been developed and is currently being implemented.

Usage Records are being prepared and submitted to the Logger service by the Grid Manager. By caching URs on the resource front-end, sufficiently reliable message delivery is obtained.

Although logging by definition is a centralized service, the target Logger service where the UR is submitted can be selected by both resource administrators via the Grid Manager configuration and by Grid users via their Grid job description. A Usage Record of a Grid job can be submitted to multiple Logger Services, multiple Loggers can coexist and be simultaneously used. This makes it possible to have Logger service deployed per infrastructure or per VO or for a user to have a personal Logger.

To query the information stored in the Logger database ARC provides a command line tool. Besides the command line tool, the NGLogger Web interface [13] was developed for convenient access to the URs stored in the database. Items like number of various kind of jobs per cluster or per user can be seen in a textual and graphical way.

6.3 Application Portals

As more research areas need more computational resources, it is very important to provide easy access to these resources, both via Web based interfaces and from the command line. Currently, most users of the computational resources use command line tools for accomplishing tasks. For other scientific areas this way of accessing a system can be very unfamiliar. To expand the user base at the center for scientific and technical computing in Lund (Lunarc [49]), an application portal project was initiated. The projects aims to provide a Web based interface to the most commonly used applications, such as MOLCAS [50], ABAQUS [51], MATLAB [52] and Python.

Implementation of the Lunarc Application Portal is designed to be lightweight and easily extendable. An important aspect of this is not to reinvent the wheel. Existing software packages and tools are used as much as possible. The architecture is illustrated in Figure 8. The Lunarc Application Portal is implemented in Python using the WebWare application server [53]. WebWare is an object-oriented application server capable of handling multiple user Web sessions. Each page on the server is represented by a Python class. The application server is integrated into the Apache Web server [54] using the `mod_webkit` Apache module. Job submission and management is handled using the ARC middleware. The current version of the portal uses the command line tools to interface with ARC, but the development version of the portal will use ARCLib.

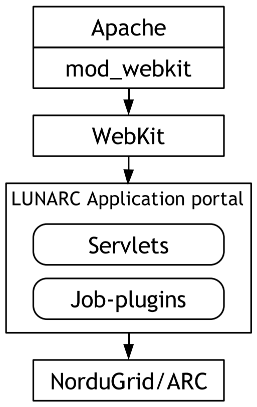

Fig. 8. Application Portal architecture

7 Discussion, conclusions and future work

In this paper, we have presented NorduGrid collaboration's Advanced Resource Connector (ARC) middleware, starting from the user requirements and proceeding to the architectural and software engineering related solutions that were implemented based on the requirements.

In order to meet the requirement of performance, security and easy deployment, the corner-stones of ARC on the server side are the *Information System*, built on LDAP, that provides a fast access to the information of computing and storage enviroment; the *Grid Manager-GridFTP* pair offering a powerful and versatile computing service and *storage services*. On the client side, there is a job submission client that implements matchmaking between the job requirements and the resources described by the Information System. The standalone client package, available to a variety of Unix platforms, is easy to install and use.

The ARC middleware has quickly grown from a prototype to a production-quality solution allowing quick and easy setup of a robust and reliable Grid computation infrastructure. In order to simplify the task of migrating users to the new environment, the developers offer an excellent community support via ticketing system and mailing lists, comprehensive documentation, and easy to download and install client packages. The software, documentation, access to Web based tools etc are made available by the NorduGrid [4].

Since there are several solutions on the market offering Grid services, standardization and consequent interoperability between them becomes as essential as interoperability of electrical appliances on the power grid — and even more so, as user jobs have much higher mobility.

ARC as a second generation Grid middleware is based on standard libraries and solutions provided by Globus®, OpenSSL, OpenLDAP and VOMS; and offers public interfaces through standard channels: GridFTP and LDAP. Nevertheless, interoperability among second generation Grid middlewares (like LCG [55], gLite [19], Globus [2] and Unicore [56]) is not easily obtainable. The gradual development and acceptance of Grid standards such as Open Grid Service Architecture OGSA [57] and Web Service Resource Framework WS-RF [58] will hopefully bring us closer to an ideal situation where users would not have to care which Grid client to use. In the future, communication with Grid servers that implement such functionalities as resource discovery, job submission, data management etc, will be available through standard interfaces and protocols. With ARC, a Web service based interface (using its own HTTPg engine [59]), is being developed. Another area of standardization is that of job submission. JSDL [60] is a forthcoming standard of an XML based job submission description language. It is foreseen that ARC will use JSDL in the near future.

With these efforts under way, ARC is gradually converging to the community standards set by the Global Grid Forum and similar bodies.

References

[1] M. Litzkow, M. Livny, M. Mutka, Condor - A Hunter of Idle Workstations, in: Proc. of the 8th International Conference of Distributed Computing Systems, 1988, pp. 104–111.

[2] I. Foster, C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications 11 (2).

[3] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, A Security Architecture for Computational Grids, in: Proceedings of the 5th ACM Conference on Computer and Communications Security, San Francisco, California, USA, 1998.

[4] The NorduGrid Collaboration, Web site.
URL `http://www.nordugrid.org`

[5] M. Ellert, A. Konstantinov, B. Kónya, O. Smirnova, A. Wäänänen, The NorduGrid project: Using Globus toolkit for building Grid infrastructure, Nucl. Instr. and Methods **A** 502 (2003) 407–410.

[6] F. Ould-Saada, The NorduGrid Collaboration, SWITCHjournal 1 (2004) 23–24.

[7] Estonian Grid, Web site.
URL `http://grid.eenet.ee`

[8] SWEGRID: the Swedish Grid testbed, Web site.
URL `http://www.swegrid.se`

[9] Danish Center for Grid Computing, Web site.
URL `Available at: http://www.dcgc.dk`

[10] S. Gadomski, C. Häberli, F. Orellana, G. L. Volpato, The Swiss ATLAS Computing Prototype, Tech. Rep. CERN-ATL-COM-SOFT-2005-007, ATLAS note (2005).

[11] P. Eerola, B. Kónya, O. Smirnova, T. Ekelöf, M. Ellert, J. Hansen, J. Nielsen, A. Wäänänen, A. Konstantinov, F. Ould-Saada, Building a Production Grid in Scandinavia, IEEE Internet Computing 7 (4) (2003) 27–35.

[12] P. Eerola, B. Kónya, O. Smirnova, T. Ekelöf, M. Ellert, J. Hansen, J. Nielsen, A. Wäänänen, S. Hellman, A. Konstantinov, T. Myklebust, F. Ould-Saada, Atlas Data-Challenge 1 on NorduGrid, in: Proc. of CHEP 2003, La Jolla, California, 2003.

[13] B. Kónya, P. Eerola, T. Ekelöf, M. Ellert, J. Hansen, A. Konstantinov, J. Nielsen, F. Ould-Saada, O. Smirnova, A. Wäänänen, U. Erkarslan, K. Pajchel, Usage Statistics and Usage Patterns on the NorduGrid: Analyzing the Logging Informations Collected on One of the Largest Production Grids in the World, in: A. Aimar, J. Harvey, N. Knoors (Eds.), Proc. of CHEP 2004, CERN-2005-002, Vol. 1, 2005, p. 711.

[14] The ATLAS Collabiration, ATLAS - A Toroidal LHC ApparatuS, Web site.
URL `http://atlas.web.cern.ch`

[15] Nordic universities network, Web site.
URL `http://www.nordu.net`

[16] H. Stockinger, A. Samad, K. Holtman, B. Allcock, I. Foster, B. Tierney, File and object replication in data grids, Cluster Computing 5 (3).

[17] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripenu, B. Schwartzkopf, H. Stockinger, K. Stockinger, B. Tierney, Giggle: A framework for constructing scalable replica location services, in: Proceeding of IEEE Supercomputing 2002, 2002.

[18] P. Kunszt, Fireman catalog user guide, Web site.
URL `https://edms.cern.ch/document/570780`

[19] EGEE gLite, gLite – Lightweight Middleware for Grid Computing, Web site.
URL `http://glite.web.cern.ch/glite/`

[20] R. Alfieri, R. Cecchini, V. Ciashini, L. dell'Agnello, A. Frohner, K. Lorentey, F. Spataro, VOMS an authorization system for virtual organizations, in: Proc. of the 1st European Across Grids Conference, Santiago de Compostela, Spain, 2003.

[21] S. Shirasuna, A. Slominski, L. Fang, D. Gannon, Performance Comparison of Security Mechanisms for Grid Services, in: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, 2004.

[22] G. Aloisio, M. Cafaro, D. Lezzi, R. van Engelen, Secure Web Services with Globus GSI and gSOAP, in: H. Kosch, L. Böszörményi, H. Hellwagner (Eds.), Euro-Par 2003 Parallel Processing: 9th International Euro-Par Conference Klagenfurt, Austria, August 26-29, 2003 Proceedings, Springer, 2003, Lecture Notes in Computer Science.

[23] R. L. Henderson, Job Scheduling Under the Portable Batch System, in: D. G. Feitelson, L. Rudolph (Eds.), Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, no. 949, Springer, 1995, Lecture Notes In Computer Science.

[24] W. Gentzsch, Sun Grid Engine: Towards Creating a Compute Power Grid , in: Proceedings of the 1st International Symposium on Cluster Computing and the Grid, 2001.

[25] Runtime Environment Registry, Web site.
URL `http://www.csc.fi/grid/rer/`

[26] The Gridsite Project, GridSite: Grid Access Control Language, available at: http://www.gridsite.org/1.0.x/gacl.html (2003).

[27] Storage Resource Management Working Group, Web site.
URL `http://sdm.lbl.gov/srm-wg/`

[28] OpenLDAP, Web site.
URL `http://www.openldap.org`

[29] T. A. Howes, M. C. Smith, A Scalable, Deployable Directory Service Framework for the Internet, Tech. Rep. 95-7, CITI (Center for Information Technology Integration), University of Michigan, USA (1995).

[30] M. Smith, T. A. Howes, LDAP : Programming Directory-Enabled Applications with Lightweigt Directory Access Protocol, Macmillan, 1997.

[31] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke, A Directory Service for Configuring High-performance Distributed Computations, in: IEEE International Symposium on High Performance Distributed Computing, IEEE Press, 1997.

[32] K. Czaijkowski, et al., Grid Information Services for Distributed Resource Sharing, in: IEEE International Symposium on High Performance Distributed Computing, IEEE Press, 2001.

[33] B. Konya, The NorduGrid/ARC Information System.
URL `http://www.nordugrid.org/documents/arc_infosys.pdf`

[34] J. M. Schopf, M. D'Arcy, N. Miller, L. Pearlman, I. Foster, C. Kesselman, Monitoring and Discovery in a Web Services Framework: Functionality and Performance of the Globus Toolkit's MDS4, Tech. Rep. ANL/MCS-P1248-0405, Argonne National Laboratory (2005).

[35] A. W. Cooke, et al., The Relational Grid Monitoring Architecture: Mediating Information about the Grid, Journal of Grid Computing (2).

[36] The Globus Alliance, The Globus Resource Specification Language RSL v1.0, Web site.
URL `http://www-fp.globus.org/gram/rsl_spec1.html`

[37] The NorduGrid Collaboration, Extended Resource Specification Language, available at: http://www.nordugrid.org/documents/xrsl.pdf (2002).

[38] A. Konstantinov, Protocols, Uniform Resource Locators (URL) and Extensions Supported in ARC.
URL `http://www.nordugrid.org/documents/URLs.pdf`

[39] O. Smirnova, The Grid Monitor (2002).
URL `http://www.nordugrid.org/documents/monitor.pdf`

[40] NorduGrid Web site, The Grid Monitor, the interface.
URL `http://www.nordugrid.org/monitor`

[41] D. Thomas, et al., Professional PHP4, Wrox Press, 2004.

[42] D. Beazley, SWIG and Automated C/C++ Scripting Extensions, Dr. Dobb's Journal.

[43] M. Lutz, Programming Python, O'Reilly, 2001.

[44] G. von Laszewski, J. Gawor, P. Lane, N. Rehn, M. Russell, Features of the Java Commodity Grid Kit, Concurrency and Computation: Practice and Experience 13 (8-9).

[45] The Mozilla Directory (LDAP) SDK Project, Web site.
URL `http://www.mozilla.org/directory/`

[46] A. Shoshani, Storage Resource Managers: Middleware components for Grid Storage, in: Proc. of the Nineteenth IEEE Symposium on Mass Storage Systems, 2002.

[47] J. Novotny, S. Tuecke, V. Welch, An Online Credential Repository for the Grid, in: Proc. of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, 2001.

[48] Global Grid Forum, Web site.
URL `http://www.ggf.org`

[49] Lunarc - Center for Technical and Scientifc Computing Lund University, Web site.
URL `http://www.lunarc.lu.se`

[50] Molcas, Web site.
URL `http://www.teokem.lu.se/molcas/`

[51] ABAQUS, Inc., ABAQUS suite of software for finite element analysis, Web site.
URL `http://www.hks.com/`

[52] The MathWorks, Inc., Matlab, Web site.
URL `http://www.mathworks.com`

[53] Python Web Application Toolkit, Web site.
URL `http://www.webwareforpython.org`

[54] The Apache HTTP Server Project, Web site.
URL `http://www.apache.org`

[55] CERN, The LHC Computing Grid Project - LCG, Web site (2003).
URL `http://lcg.web.cern.ch/LCG/`

[56] D. Erwin (Ed.), Unicore Plus Final Report - Uniform Interface to Computing Resources, Unicore Forum, Forschungszentrum Julich, 2003.

[57] I. Foster, C. Kesselman, J. Nick, S. Tuecke, The physiology of the Grid, in: F. Berman, G. Fox, T. Hey (Eds.), Grid Computing, John Wiley & Sons Ltd., 2003, pp. 217–250.

[58] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, The WS-Resource Framework, Version 1.0, whitepaper, available at http://www-128.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.

[59] A. Konstantinov, The HTTP(s,g) And SOAP Framework.
URL `http://www.nordugrid.org/documents/HTTP_SOAP.pdf`

[60] JSDL-WG, Job Submission Description Language Work Group, Web site. URL https://forge.gridforum.org/projects/jsdl-wg