LHC Computing Grid Project

ARCHITECTURAL ROADMAP TOWARDS DISTRIBUTED ANALYSIS

Document identifier:	**LHC-SC2-xx-2003**
Date:	**31-Oct-2003**
Authors:	**ALICE:** **P. Buncic (CERN/IKF Frankfurt)** **F. Rademakers (CERN)** **ATLAS:** **R. Jones (U.Lancaster)** **R. Gardner (U.Chicago)** **CMS:** **L.A.T. Bauerdick (Fermilab), chair** **L. Silvestris (INFN Bari)** **LHCb:** **P. Charpentier (CERN)** **A. Tsaregorodtsev(IN2P3 Marseille)** **LCG:** **D. Foster (CERN)** **T. Wenaus (BNL)** **GAG:** **F. Carminati (CERN)**
Document status:	**v1.00**

Abstract:

This document is the report of the LHC Computing Grid Project's Requirements and Technical Assessment Group (RTAG) on an “Architectural Roadmap for Distributed Analysis” (ARDA).

CONTENTS

1 INTRODUCTION

This document is the report of the LHC Computing Grid Project's Requirements and Technical Assessment Group (RTAG) on an "Architectural Roadmap for Distributed Analysis" (ARDA).

For this roadmap we recommend a series of waypoints. This document represents the first of these, namely the findings and recommendations of the ARDA RTAG. The RTAG reviewed existing projects with the aim to capture their architecture in a consistent way, confronting them with the Hepcal-II use cases[1], in order to come to an initial blueprint of an architecture for distributed analysis at the LHC. Following the guidance of the mandate given by the SC2, ARDA defines this architecture in terms of a set of collaborating Grid services with well-defined interfaces. We propose as the next step to build a prototype of the ARDA architecture.

1.1 EXECUTIVE SUMMARY

The Hepcal-II document [1] describes a set of use cases for analysis as an iterative sequence of user interactions. In general these consist of a series of steps, starting with a *setup stage*, where the user authenticates to the system, which in turn creates the analysis context and ensures the correct installation of software packages, *formulation of input and algorithm***,** where the datasets are selected through metadata queries and algorithms are specified, *execution of selection and algorithms*, while the user can monitor progress and eventually look at intermediate results, and finally *gathering the results*, passing resulting datasets back to the session context where they are stored and can be published and shared.

This model requires persistency of the analysis workspace associated with a given analysis task, that a user can later re-connect to, re-submit the analysis with modified parameters or code, check the status, merge results between analyses, share datasets with other users and analysis workspaces, while the system keeps provenance information about jobs and datasets.

The ARDA RTAG reviewed existing projects, aiming to capture their architecture in a consistent way, confronting them with these use cases where possible. Several of these projects address distributed analysis by providing a set of web services or intelligent agents that communicate through a well-defined protocol. ARDA adopted that approach and studied a decomposition of the distributed analysis system into a set of services that would implement the major use cases and provide the basic required functionality for distributed physics analysis. We found that of the projects surveyed AliEn [2] had the most complete distributed analysis functionality, implemented through a set of web services. It addressed a large part of the aforementioned use cases and had a substantial history of successful use.

The next step for the RTAG was to derive a decomposition of the distributed analysis environment into a web services description, and to define their functions and interfaces. For this the RTAG started from the AliEn web services, re-factoring and synthesizing with the input gained from the other projects examined. The proposed domain decomposition into the ARDA set of services is described in this document.

The RTAG then went back to the other Grid projects related to distributed analysis, to study how these fit into the ARDA decomposition and what other and possibly complementary services are provided addressing distributed analysis functionality.

As a result, the RTAG defines the ARDA architecture as a set of services, with specific behavior and interfaces. Proceeding the way described allowed the RTAG to base its recommendations on an analysis and synthesis of several projects, with at least one existing implementation. The resulting ARDA blueprint is presented in this document.

1.1.1 General Recommendations

We recommend that the ARDA architecture should be implemented as an OGSI [3] compliant set of services that implement the distributed analysis functionality. Being OGSI compliant, the ARDA services would inherit industry-standard services architecture and a common set of functionalities, like communication protocols, lifetime support, service compositions, etc. We also recommend that a

prototype of such a system be developed and deployed rapidly, to allow users to gain experience with it and to provide feedback on functionality and interfaces.

We do however note that any implementation of OGSI, including the Globus Toolkit version 3 (GT3) [4] is new technology, whose reliability, resilience and performance is not proven and needs to be studied in at-scale environments. We recommend that the LCG address this upfront, using modeling, performance studies in test environments, by developing a plan for scaling up to LHC workloads, and by engineering of the underlying services infrastructure.

The ARDA services should present an Grid Access Service Factory and Application Programming Interfaces (GAS/API) to enable applications, analysis shells, experiment frameworks etc to interface to the distributed analysis services through a well defined API, with bindings to a required set of programming languages. Instantiating the ARDA GAS/API service would provide a session context that handles the user authentication, state, etc. The API allows to interface the experiment's frameworks and analysis shells, and to build web portals and other user interfaces to the distributed analysis environment through a well-defined common interface.

The API should provide methods for file access, submission and following up of the jobs, , meta-data access, file catalog access etc. Higher level analysis services can be implemented on top of the ARDA layer, either by going through the GAS/API or by communicating directly with the ARDA services.

We recommend a common API to the distributed Grid environment for analysis, shared by all LHC experiments. This would be a vehicle for developing common functionality between experiments, and at the same time allow development and integration of experiment specific services. For example, while it is anticipated that the interactions on event-level metadata will be quite different for different experiments, it is expected that this could be encapsulated in the interactions between experiment specific metadata catalog service and common file catalog service. The OGSI specification adopts a web service component model to address such issues.

In order to implement the distributed analysis use cases, some of the ARDA services will be statefull and persistent. We recommend that an initial implementation should provide this through a database backend.

1.1.2 Recommendations for an ARDA Prototype

As a next waypoint on the ARDA roadmap we recommend to develop and deploy an initial prototype of the ARDA architecture, with the main goal to provide a more complete blueprint and to develop the specifications for functionality and interfaces of the ARDA services and API.

The ARDA prototype would allow the investigation of the LHC requirements for distributed analysis further. We recognize the value of real prototyping of services, their functions and interfaces, providing insights how the services implement the required functionality for distributed analysis. The prototype will allow the realistic investigation of the possible commonality between the experiments in the API, which is ultimately necessary for the LHC experiments to run their competitive physics research on top of a shared multi-VO environment.

It has been pointed out that there is no “evolutionary” path from the current GT2-based[5] LCG infrastructure, building upon the VDT[6] and EDG[7] components and software stack, to a Grid services architecture based on OGSI compliant services model. The prototype addresses this as it enables one to perform real-world OGSI modeling, functionality and performance tests and to address the issue of how to deploy and run ARDA services along with the existing ones on the LCG-1 resources. There are several areas where the ARDA services would leverage the emerging experience with running a large distributed environment and exploit the R&D in the Grid middleware projects, e.g. in the area of VO management, security infrastructure, set-up of the Computing and Storage Elements and interfacing to the fabrics etc.

We recommend that the LCG setup a project to develop the prototype, considering these main goals. The schedule and milestones should be commensurate with the need to expose the resulting functionality and interfaces early to the community, and be on a timescale of 6 months.

Grid services descriptions and Grid services instances should be defined early in the project. This should include descriptions and specification of core ARDA services, the definition of the external interfaces, that is, the API to experiment frameworks, services and tools, and the interfaces to infrastructure, facilities and fabrics. It is here where experiment developers and other Grid efforts should be engaged from the beginning, and an initial release of the ARDA prototype should be made available to the experiments and the Grid community. The internal interfaces between services should be defined and documented with the end of the prototyping phase of about 6 months. Specific points of interactions with the community should be provided, through early releases, workshops, and actively seeking feedback on API, service interfaces and functionalities.

We recommend that the ARDA prototype project start with a careful definition of the work areas. The constituency of the project and the project lead should be identified quickly, so that a team can be built. The project should develop and present the work plan, schedule and milestones, including a plan for interfacing to and engaging of LHC experiments and the LHC-related Grid community.

We propose a four-prong approach towards these goals:

1) Re-factoring of AliEn and possibly other services into ARDA, with a first release based on OGSI::Lite[8]; consolidation of the API working with the experiments and the LCG-AA; initial release of a fully functional prototype. Subsequently implementation of agreed interfaces, testing and release of the prototype implementation.
2) Modeling of an OGSI-based services infrastructure, performance tests and quality assurance of the prototype implementation
3) Interfacing to LCG-AA software like POOL and ROOT
4) Interfacing to experiment's frameworks, with specific meta-data handlers and experiment specific services

The LCG Application Area, the LCG Grid Technology Area, and the EGEE[9] middleware team should be involved in the ARDA. We recommend the experiments be involved from the beginning by working on the interface with the frameworks and integrating into the experiment's data and metadata management environments. Other LHC-related projects should be engaged by exposing services and GAS/API definitions early to allow synergistic developments from these projects. It is important that the appropriate emphasis and effort is being put on documentation, packaging, releases, deployment and support issues.

1.1.3 Beyond the ARDA Prototype

After the prototype phase, we expect a period of scaling-up and re-engineering the OGSI foundation, the database, the information services etc.; deployment and interfaces to site and Grid operations, VO management, information services infrastructure etc.; building higher level services and experiment specific functionality; work on interactive analysis interfaces and new functionalities.

The list of ARDA services discussed in this document is by no means complete. It is likely that a fully functional system will require additional services, for example management of virtual data, handling of experiment specific services tied to persistency and event access models, specific VO policy services or high level optimization and supervision services.

The ARDA blueprint offers the basic functionality required to fulfill the common needs of the LHC experiments, while allowing building upon the basic services described in this document. This approach provides opportunity for well-aligned further developments and the inclusion of new and advanced functionalities.

The rest of the document is structured as follows:

Section 2 gives a high-level view on the requirements for distributed analysis on the Grid, as extracted from HEPCAL-II. We then provide summary descriptions of some of the projects developed for the distributed analysis in different experiments in Section 3. In Sections 4 and 5 we give a description of the ARDA blueprint and the scope and proposed functionality of an ARDA prototype.

1.2 SC2 MANDATE TO THE ARDA RTAG

Observation:

Different LHC experiments have developed packages (AliEn, Ganga, Dirac, Impala, Boss, Grappa, Magda…) that either sit on top, complement, expand or parallel the functionality of the Grid middleware (VDT, EDG…). At this time the LCG is coming to grips with the middleware development requirements. There is an expectation that OGSA Services Architecture will be the basis for future development. The Experiments need to specify in their TDR's, baselines, fallback and development strategies.

Motivation:

- To agree on requirements as laid out in a first step by recent work within the GAG and identify commonalities within the current projects that might allow the LCG (both in the AA and GTA areas) to provide a focus of effort.
- To provide guidance to the LCG on future Middleware development directions and interfacing work to match the experiment requirements
- To build on the richness of the current technical solutions to avoid duplication of efforts
- To clearly identify the roles and responsibilities of the components/layers/ services in the experiment DA planning
- To give guidance to the community on the expected division of work between the experiments, the LCG and the external projects.

Mandate:

- To review the current Distributed Analysis (DA) activities and to capture their architectures in a consistent way
- To confront these existing projects to the HEPCAL II use cases and the user's potential work environments in order to explore potential shortcomings
- To consider the interfaces between Grid, LCG and experiment-specific services
    - Review the functionality of experiment-specific packages, state of advancement and role in the experiment
    - Identify similar functionalities in the different packages
    - Identify functionalities and components that could be integrated in the generic Grid middleware
- To confront the current projects with critical Grid areas

To develop a roadmap specifying wherever possible the architecture, the components and potential sources of deliverables to guide the medium term (2 year) work of the LCG and the DA planning in the experiments

Schedule

The RTAG shall provide a draft report to the SC2 by September 12. It should contain initial guidance to the LCG and the experiments to inform the September LHCC manpower review, in particular on the expected responsibilities of

- The experiment projects
- The LCG (Development and interfacing work rather than coordination work)
- The external projects

The final RTAG report is expected for October 03, 2003.

Makeup

The RTAG shall be composed of

- Two members from each experiment

- Representatives of the LCG GTA and AA

If not included above, the RTAG shall co-opt or invite representatives from the major Distributed Analysis projects and non-LHC running experiments with DA experience.

1.3 RTAG ACTIVITIES

The ARDA RTAG has held meetings about once a week from July to October 2003. The RTAG program started with initial presentations from the following projects related to distributed analysis: PROOF[16], AliEn[2], DIAL[10], Clarens[11], GANGA[12], DIRAC[13] and a two-days workshop in September, leading to the initial presentation to the SC2 on September 12, 2003[14].

After that, members of ARDA had extensive discussions with their experiments and the LCG, and got feedback on the initial report.

A verbal final report was given to the SC2 on October 3, 2003[15].

2 REQUIREMENTS FOR DISTRIBUTED ANALYSIS ON THE GRID

The LCG Grid Applications Group undertook a detailed review of the distributed analysis requirements recently [1]. In the following we give a brief summary of the analysis activity that must be supported by the functionality of the eventual Grid system. A high level scenario of the analysis activity can be summarized as the following.

A user or a group of users have a (set of) algorithm(s) that they want to apply to a particular selection of input data in a given execution environment. The input data are selected via a query to a **Metadata Catalogue**. The selection and the algorithms are passed to a **Workload Management** system together with the specification of the execution environment. The algorithms are executed on one or many worker nodes of a **Computing Element**. During this execution, they need to access physical data datasets (DS) usually on a local **Storage Element**, using a **DatasetCatalog**. The user **monitors** the progress of the job execution. The results are gathered together and passed back to the owner of the job. The resulting datasets are stored and can be published in order to be accessible for other users.

The above scenario represents the analysis activity from the user perspective. However, some other actions are done behind the scene of the user interface:

- To carry out the analysis tasks users are accessing shared computing resources. To do so, they must be registered with their Virtual Organization (VO), **authenticated** and their actions must be **authorized** according to their roles within the VO; the consumption of the computing resources by the users is **accounted** for;
- The user specifies the necessary execution environment (software packages, databases, system requirements, etc) and the system insures it on the execution node. In particular, the necessary environment can be **installed** according to the needs of a particular job;

The execution of the user job may trigger some **Data Management** actions such as transfers of files or datasets between a user interface computer, execution nodes and storage elements. These transfers are meant to be transparent for the user.

In the above schematic description of a typical analysis sequence, it appears clear that the user interacts with a certain number of loosely coupled components or services. These components are highlighted in the text in **bold**. They can be roughly classified in the following categories:

- User Interface

  It helps the user to prepare his or her job, submit it, monitor it and retrieve its results. The user interface is typically the experiment framework, Web portal or a command line interface.

- Virtual Organisation Services

  These services allow for authentication, authorization and accounting within a given set of users (usually a collaboration)

- Data Management
  - Metadata information: allows users to query datasets based on various information such as the type and origin of the data (data taking period, previous processings) etc. Datasets are therein referred to by their unique identification tag without reference to their physical location.
  - Dataset catalog information: allows replication of datasets onto several sites. Multiple instances of datasets at various sites as well as generic dataset information (file size, technology…) may be kept as well. Logical names are usually associated to datasets in order to facilitate naming, as dataset identification such as GUID are not practical.
  - Data Management tools: allows dataset replication, replica deletion, and more generally any dataset manipulation based on the Dataset Catalog information.

- Workload Management

  Principally dealing with the whole process from the job submission to the start of execution on a worker node. In particular, it has to ensure that the system requirements

defined in the job description are met, that the necessary input datasets are accessible from the selected worker node (possibly using the Data Management tools). It allows the follow-up of the job until its completion as well as the replication of output datasets depending on the job specification.

- Miscellaneous needs

  For each dataset, some information might need to be kept that is not meant for being queried when defining a dataset (e.g. identification of the worker node that produced it, operating system, CPU time used…). This is referred to as **Provenance** of the dataset.

It is useful to be able to have an overview of the status of available resources (e.g. number of available nodes at each site, number of jobs pending, running…). This constitutes a means for **Grid Monitoring**.

3 SUMMARY OF GRID PROJECTS PRESENTED TO THIS RTAG

3.1 PROOF: PARALLEL ROOT FACILITY

The Parallel ROOT Facility[16], PROOF, is an extension of the ROOT system [17] that allows interactive and transparent analysis of very large sets of ROOT files in parallel on remote computer clusters. The main design goals for the PROOF system are transparency, scalability and adaptability. Transparency means that there should be as little difference as possible between a local ROOT based analysis session and a remote parallel PROOF session, both being interactive and giving the same results. Scalability means that the basic architecture should not put any implicit limitations on the number of computers that can be used in parallel. Adaptability means that the system should be able to adapt itself to variations in the remote environment, e.g. changing load on the cluster nodes, network interruptions, etc.

Being an extension of the ROOT system, PROOF is designed to work on objects in ROOT data stores. These objects can be individually `keyed` objects as well as **`TTree`** based object collections. By logically grouping many ROOT files into a single object, very large data sets can be created. In a local cluster environment these data files can be distributed over the disks of the cluster nodes or made available via a NAS or SAN solution.

By employing Grid middleware, PROOF is extended from single clusters to a virtual global cluster. In such an environment the processing may take longer (not interactive), but the user will still be presented with a single result, like the processing was done locally.

3.1.1 System Architecture

PROOF consists of a three-tier architecture, the ROOT client session, the PROOF master server and the PROOF slave servers. The user connects from his ROOT session to a master server on a remote cluster and the master server in turn creates slave servers on all the nodes in the cluster. All the slave servers process queries in parallel. Using a pull protocol the slave servers ask the master for work packets, which allows the master to distribute customized packets for each slave server. Slower slaves get smaller work packets than faster ones and faster ones process more packets. In this scheme the parallel processing performance is a function of the duration of each small job, packet, and the networking bandwidth and latency. Since the bandwidth and latency of a networked cluster are fixed the main tuneable parameter in this scheme is the packet size. If the packet size is chosen too small the parallelism will suffer due to the communication overhead caused by the many packets sent over the network between the master and the slave servers. If the packet size is too large the effect of the difference in performance of each node is not evened out sufficiently. This allows the PROOF system to adapt itself to the performance and load on each individual cluster node and to optimise the job execution time. Performance measurements show a very good scalability and efficiency.

3.1.2 PROOF and the Grid

To be able to build a global virtual PROOF cluster Grid services need to be used as described in this document. The interfacing of PROOF to the Grid can be done at several levels. The following levels have been identified:

- Interface to the Grid file catalogue allowing a user to select a data set based on tags or logical file names (using wildcards etc).
- Interface to the Grid resource broker to find the best location(s), based on the data set, where to run the query. This could trigger the replication of some missing files to a cluster (only when the amount of data in the files is relatively small).
- Interface to the Grid job queue manager to start PROOF master and slave daemons on the remote cluster. The ROOT client will then connect to these pre-started daemons to create a

full PROOF session. This will require the grid queuing system that supports interactive high priority jobs.

The PROOF teams is currently working with the AliEn developers on a prototype that implements these features, where the AliEn API is accessed via the abstract TGrid class from ROOT.

3.2 ALIEN: ALICE ANALYSIS ENVIRONMENT

3.2.1 Scope and expected deliverables

AliEn is a distributed production environment developed by ALICE that represents a self-contained Grid-like system based on Web Services technology [18]. It is meant to satisfy the principal HEP use cases (simulation, reconstruction and analysis) in a distributed and heterogeneous environment where the data component plays an important role (large number of big, read only files containing one or more physics events at various processing stages).

3.2.2 Technologies used

The system is built around Open Source components and uses standard protocols (SOAP) to implement a distributed computing platform that is currently being used to produce and analyse Monte Carlo data at over 30 sites. The near term goal is to make AliEn services compatible with the OGSI model that has been proposed as a common foundation for future Grids.

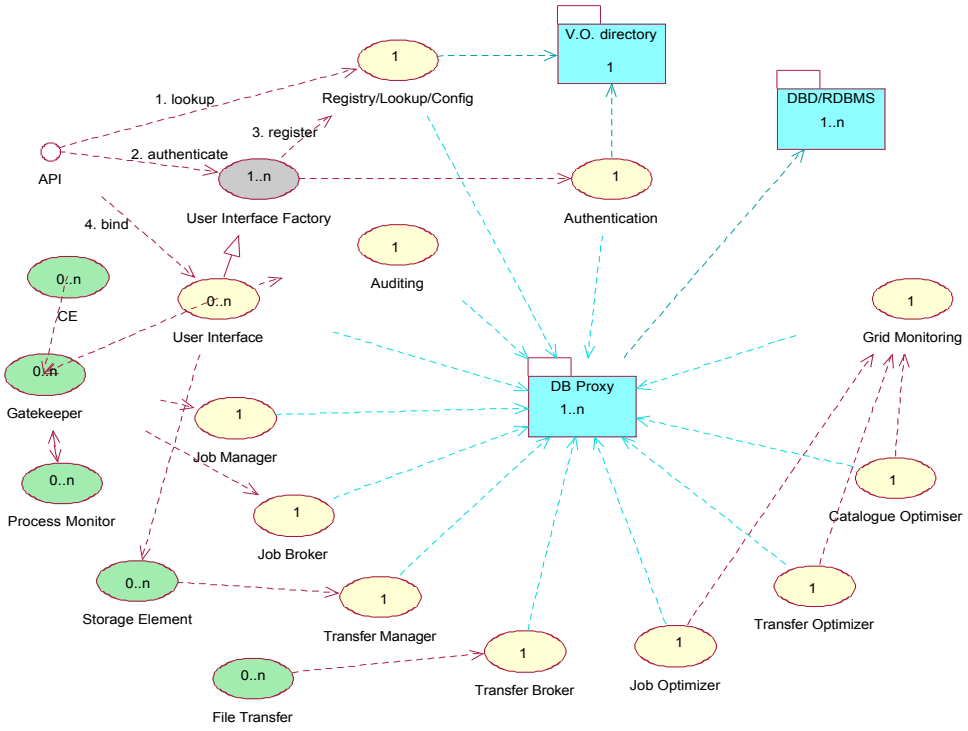

Figure 1: This diagram shows currently implemented Web Services in the AliEn framework and their interactions while executing a typical analysis use case.

The backbone of AliEn is a distributed relational database (currently, but not necessary, MySQL). The users must present a valid Globus proxy certificate, RSA key or password to the Authentication Service in order to receive a session token, which then can be used to gain access to the database via a database Proxy Service. In this model, users own their partition of the database and the database engine is used to impose strict access control, authorisation and eventually auditing. Each user

belongs to a VO and can have multiple roles in the system. AliEn VO management and service configuration management is based on a LDAP directory.

3.2.3 System Architecture

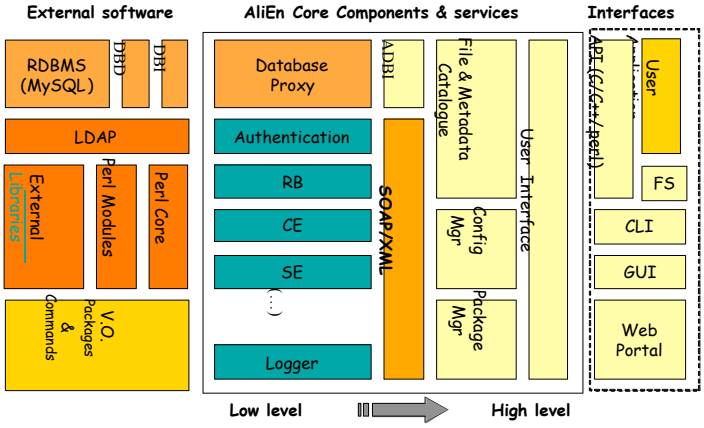

Figure 2: The building blocks of AliEn

The system is built around Open Source components that represent about 95% of the code. The core AliEn components and services are implemented in the perl scripting language. A portion of the database represents a File Catalogue, which associates a universal logical file name with physical file name(s) for each dataset and provides transparent access to datasets independently of their physical location. Besides regular files, the File Catalogue supports virtual files linked to job status and provenance information (job id, stdout/stderr, job input and output files).

Logical files can be further annotated on a per-directory basis by tagging them with additional attributes accessible via an extended File Catalogue interface, implementing a Metadata Catalogue.

A user submits a job to the Workload Management, which consist of Job Broker, Manager and Optimizer services. Requirements on the job are expressed in JDL, based on Condor ClassAds[19], and once all these requirements are satisfied on at least one of the Computing Elements, the job will be reserved by that CE first reporting free capacity to the Job Broker. In AliEn, the Job Broker is implemented using a simplified pull (as opposed to a traditional push) model. Optimizer services make sure that job requirements are eventually satisfied, which may require triggering file replication or even execution of other jobs. The AliEn CE has been interfaced to a number of batch systems (LSB, PBS, BQS, SGE, Globus, Condor, etc). While running, a job is wrapped in another service (Process Monitor) that allows the user to interact with it. Interactions with local services running on a site happen via the Cluster Monitor service.

Upon job completion, the output files are stored at a near Storage Element, which can have interfaces to various mass storage systems (CASTOR, HPSS, HSI, etc).

File replication and file transport is carried out under the control of Data Management Services (File Transport, Transfer Manager, Transfer Broker, Catalogue Optimizer). These services work together in a way that resembles the job execution model, providing a scheduled and reliable file transport service that is can employ several transport protocols (bbftp, gridFtp).

The overall performance of the systems and services is being monitored using the MonALISA framework [20], which is currently integrated in the system as a monitoring display, but ultimately aims at active monitoring, simulation and cost evaluation functionalities.

In order to gain access to the Grid at the application level, AliEn provides an API. Besides the native perl module that allows users to use, modify or extend the interface, AliEn provides a C and C++ API. In particular, the C++ API is thread-safe and was used to implement a full-featured file system on top of the AliEn File Catalogue. The AliEn File System (alienfs) integrates the AliEn file catalogue as a

new file system type into the Linux kernel using LUFS, a hybrid user space file system framework. The AliEn framework is used for authentication; catalogue browsing, file registration and read/write transfer operations.

3.2.4 Supported analysis models

AliEn has been integrated into the ROOT and PROOF frameworks and supports both asynchronous (interactive batch) and synchronous (truely interactive) analysis models. The **`TAlien`** class, based on the abstract **`TGrid`** class, implements the basic methods to connect and disconnect from the Grid environment and to browse the virtual file catalogue. **`TAlien`** uses the AliEn API for accessing and browsing the file catalogue. The **`TAlienFile`** class extends the ROOT **`TFile`** class and provides additional file access protocols using the generic file access interface of the AliEn C++ API.

In the batch analysis model, jobs are split by a Job Optimizer taking into account data location and are executed in parallel, as soon as required resources become available. ROOT result files can be merged on demand. Analysis jobs are configured, executed and made persistent on the Grid using a new ROOT analysis class (**`TAlienAnalysis`**), which also provides the analysis framework and overall steering of the analysis task.

The interactive analysis model with AliEn extends the PROOF functionality to a multi-site setup over the wide area network. The PROOF client connects to a PROOF master server running on a AliEn core service machine. PROOF daemons on remote sites are started dynamically using dedicated queues in the site batch queue system. They are assigned on demand corresponding to the queried data set. The PROOF master connects to PROOF daemons at distributed sites through an AliEn TCP routing service. This enables connectivity and connection control to computing farms on private networks. A dedicated PROOF Grid service opens TCP routes through a site multiplexer and maintains the population of PROOF daemons corresponding to the computational needs of the user community. Results are available on the fly using a client/server architecture. The user interface remains compatible with the standard implementation of PROOF.

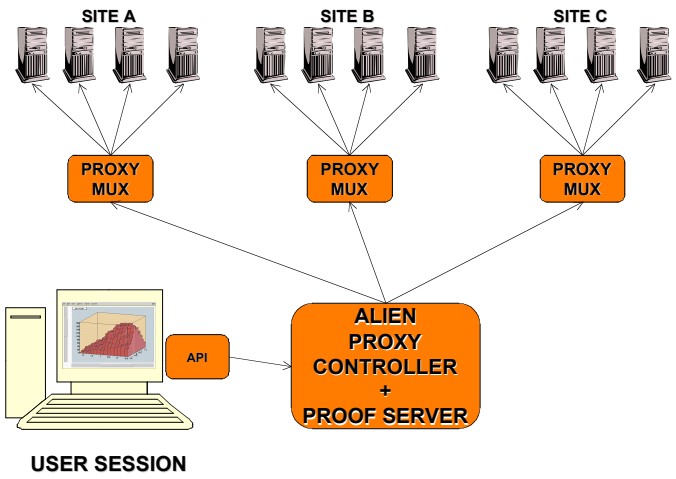

Figure 3: PROOF setup in the AliEn environment

3.2.5 State of deliverables

AliEn today is able to solve the ALICE simulation and reconstruction use cases, and tackles the problem of distributed analysis on the Grid following both approaches of asynchronous and synchronous analysis, making the distributed AliEn environment available to the ROOT prompt and using PROOF functionality.

3.3 CLARENS

Clarens [11] is a web services layer with a strong emphasis on security and distributed management. Clients and services provide the real analysis functionality. Existing services include secure file access, an interface to the SDSC Storage Resource Broker[21], the POOL file catalog, VO management, and proxy escrow. Clients include ROOT, Iguana, PDA and desktop versions of JAS[26] and the WIRED event display[22], a web interface and a Python scripting client.

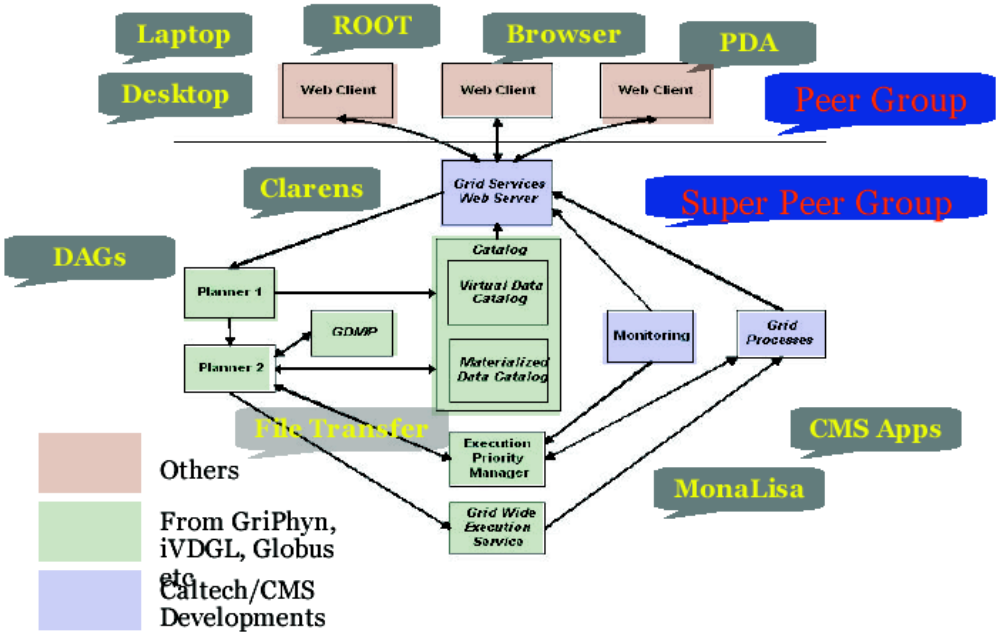

Figure 4: CAIGEE Architecture: interaction between different components

3.3.1 Supported analysis model

The CAIGEE system [23] uses Clarens as a middleware component. It is oriented towards a services model of analysis, but with support of the classical Globus-type shell command execution model. i.e. a single web services interface is exposed by mutually aware distributed servers.

3.3.2 Technologies used

For Clarens itself the Apache web server is used with extensions in Python to implement the actual web service functionality. There is also a Java version of the server being developed using the Tomcat servlet engine.

Authentication is based on PKI Grid certificates, including proxy certificates created by Globus, VOMS or Clarens. The browser client uses client-side certificate authentication over SSL. Client-server communication is through SOAP or XML-RPC, and clients do not require Globus to be installed to contact servers.

3.3.3 Scope and expected deliverables

The scope of CAIGEE is to provide a working implementation of a system that allows the interactions between components as shown in the Figure. Deliverables include the clients and server middleware to make these interactions possible, including distribution and installation of the components. More information is found in reference [23].

3.3.4 State of deliverables

The existing services and clients are described in the previous section. Current services in development include auditing (logging), server discovery and distributed file catalogs, metadata catalog to complement the POOL virtual data catalog service, third party data transfer, and an CMS ORCA/COBRA interactive remote analysis service. The University of Florida CMS group is using the Java server to provide a Clarens service of their Sphinx virtual data system[24].

Planned services include resource planning and reservation, a Grid-wide execution service, access to monitoring data provided by e.g. MonALISA[25].

3.4 DIAL: DISTRIBUTED INTERACTIVE ANALYSIS OF LARGE DATASETS

DIAL [10] is a project to investigate HEP distributed interactive analysis. Its three primary goals are: to demonstrate the feasibility of distributed analysis of large datasets; to set corresponding requirements for Grid components and services; and to provide the ATLAS experiment with a useful distributed analysis environment. It has many of the aspects of a workflow project, and the decomposition for interactive analysis also provides an effective solution for batch analysis. Prototype classes exist in C++ that have been tested with a distributed local batch system and a ROOT front-end application; PROOF and JAS[26] are also being added as front-ends.

A typical analysis user identifies data of interest, defines an algorithm for reducing that data to a result (typically a collection of analysis objects such as histograms and ntuples), uses the algorithm to process the data and generate the result, and then manipulates these objects in an analysis framework such as PAW, ROOT or JAS. DIAL concentrates on the step of processing the data to generate the result.

The DIAL team is much too small to address these issues in isolation and so has been working in the context of the PPDG Grid project[27]. A plan for integrating with GANGA has been outlined. The DIAL team looks forward to working with ARDA.

Perhaps the most important requirement to arise from the DIAL project is the need for a high-level job definition language (JDL) that enables users to frame their analysis requests and to access the generated result. DIAL defines such a language in C++ and provides an XML representation of all of the relevant objects. In the DIAL model, users interact with a "scheduler" and express their requests and receive results using this language. DIAL provides a ROOT interface to provide access to those users and has plans to provide a Python interface to allow access from GANGA and other Python-based analysis frameworks. The DIAL JDL is flexible so that any back-end application may be provided to process the data. At present DIAL supports PAW (the format of the existing data produced in ATLAS reconstruction) and there are plans to support ROOT and ATHENA. Another very important component of this JDL is the dataset used to specify the data to be processed.

DIAL has recently added a scheduler client and web service interface so that users can interact with a remote server exchanging SOAP messages. Local schedulers based on fork, LSF, lsrun and Condor are already available. The latter can also be used with Condor-G to provide (non-interactive) grid access. This is a step along the planned path to deliver a grid service early next year.

DIAL proposed to define a common high-level JDL that could be shared by different experiments and perhaps even outside of HEP. If successful, such a language would enable different experiments to share high-level schedulers in the same way they share batch systems today. Interactive analysis places very stringent requirements on such schedulers. According to DIAL, a common JDL might encourage grid middleware developers to deliver a system meeting these requirements.

3.5 GANGA: GAUDI/ATHENA AND GRID ALLIANCE

GANGA is a high level Grid Job Submission Wizard being developed in common by LHCb and ATLAS. Its aim is to help physicists to prepare, submit, monitor and retrieve jobs, including analysis jobs. The job preparation consists in the following steps:

- Select workflow to be executed (from a workflow database and/or editing)

- Prepare the execution parameters (JobOptions for Gaudi/Athena jobs); these can come from an options database and/or editing.
- Select datasets to be processed (using the Metadata catalog)
- Split the job into sub-jobs on user request in order to achieve parallel files processing
- Submit (sub-) jobs either for interactive execution, to a local batch system, to LCG or any other workload management system (e.g. Dirac)
- Monitor the progress of the jobs (possibly examine output such as logfile, histogram…)
- Retrieve jobs after completion, possibly re-submit in case of a crash
- Merge output (text files, histograms, list of output Ntuple files)

Ganga is a set of Python modules, each one dedicated to one task, connected via a Python bus. Sets of modules are of general use (job submission, monitoring, retrieval…), while others are dedicated to interface to Gaudi/Athena jobs (JobOption editor) or experiment specific (Dataset selection). Other front-end modules could be plugged-in (e.g. it has been exercised for Babar job submission, with modest effort by a Babar worker customizing the GUI).

Ganga is by no means providing Grid resources, but is using them together with experiment or framework services for helping users in painlessly interfacing to any computing infrastructure, from laptop standalone environments to Grid based infrastructures. With Ganga, users are able to test their software with exactly the same environment that will be used for analyzing large datasets.

3.6 DIRAC: DISTRIBUTED INFRASTRUCTURE WITH REMOTE AGENT CONTROL

DIRAC[13] is distributed MC Production system that fulfills the requirements for the simulation and reconstruction of the LHCb experiment. It consists of a number of central services and clients (Agents) running on each of the LHCb production sites. The central services include the following:

- Production Service is managing the job queue. It accepts jobs prepared with a web based Production Editor. It gets requests for jobs from Agents, checks the capabilities of the requesting site and serves the jobs accordingly;
- Job Monitoring Service collects and visualizes the job status information. The running jobs report their status directly to this service;
- Bookkeeping Service receives the metadata and replica information for the published datasets, stores it and serves it to its clients. It combines the functionality of a Metadata and File Catalog, in the ARDA language.

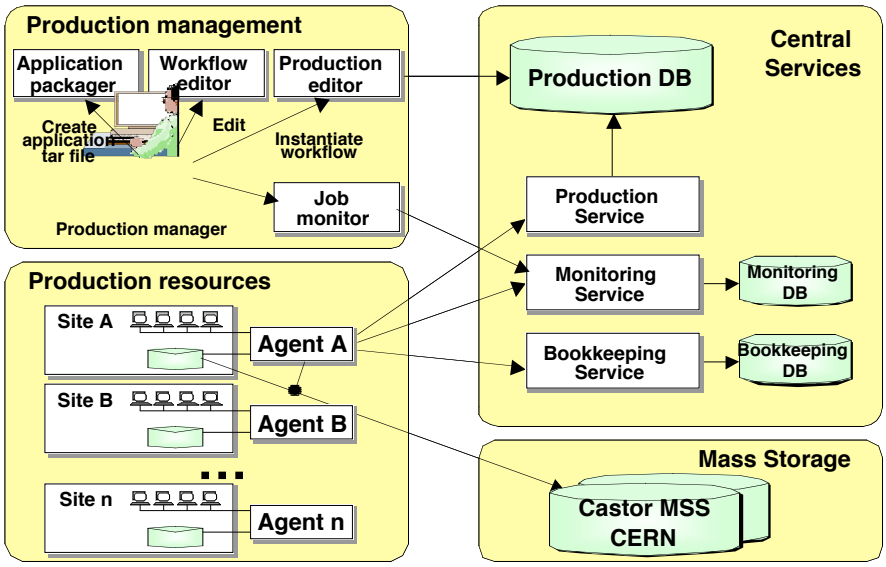

Figure 5: The DIRAC architecture

Agents running at each of the production sites monitor the status of the local batch system. If there are slots available, an Agent contacts the Production Services for the workload; gets a job and installs the execution environment (software, databases) according to its requirements; submits the job to the local batch system. The job is executed with the help of workflow executor scripts that steer the job and updates the central Job Monitoring Service with the job progress status. After the job is executed, the Agent insures the dataset transfers, updating the Bookkeeping Services with the metadata and replica information for the newly produced datasets.

The DIRAC central services and Agents are implemented as XML-RPC servers and clients. The Production, Monitoring and Bookkeeping databases are hosted by the Oracle server at CERN.

Production Service together with a set of distributed Agents form a powerful Workload Management System which realizes the “pull” scheduling paradigm. Agents are also providing the functionality of the computing resource gatekeeper (Computing Element) with back-ends for LSF, PBS, BQS, Condor and fork available. The DIRAC Workload Management System is currently being updated to cope with analysis tasks. At the same time a prototype OGSI compliant wrapping using the GT3 toolkit has been developed.

It is planned that the next generation of the DIRAC system will be based on the grid services provided in the framework of the ARDA prototype project. The DIRAC team intends to contribute its implementation of ARDA compliant Workload Management and Metadata Catalog services while acquiring other services from the ARDA development.

4 THE ARDA BLUEPRINT

As described and motivated in the introduction, the ARDA RTAG started from a careful analysis of the necessary functionality to meet the requirements for distributed analysis, based on the AliEn project that already has a substantial history of successful use. In the following we inspect the services needed to implement this functionality, with the aim to derive general domain decomposition. Confronting these services with other projects in corresponding domains and requiring that services defined by ARDA should be genuine replaceable components, we define a set of new services that must be added to the set of services derived from AliEn to complete the system. As a result of this analysis, we present an architecture defined as a minimal complete set of collaborating services with well-defined interfaces capable of satisfying the major LHC analysis use cases. We also indicate approaches to how user applications (portals, client tools, experiment frameworks, services, etc.) could be integrated to core services using ARDA Grid Service Components, such as a Grid Access Service component.

4.1 ELEMENTS OF THE COMMON GRID ENVIRONMENT

In order to organize the set of Grid services in a coherent system we need a set of rules that will allow different services to communicate with each other. The same rules adopted on the entire Grid level will allow interoperability of services. Therefore, we consider the OGSI specification by the Global Grid Forum[3] as the possible common ground for building ARDA services.

4.1.1 Service Access Protocols

The main protocol for communications between the services is that suggested by the OGSI specification[3], which extends the Web Services Definition Language (WSDL) specification[28] to incorporate stateful services, control management of long-lived process and collection of service instances. OGSI defines a component model, which extends WSDL and XML Schema, and introduces the factory and registration interfaces for discovery and creation of Grid services. WSDL includes a binding for the SOAP 1.1 messaging protocol [29].

4.1.2 Security Infrastructure

A common Security Infrastructure is an important element, as almost all the services will have security checks built in. So, the user and service credentials should be acceptable throughout the whole set of services. The GSS-API based security infrastructure (based on Globus/GSI) [30] already adopted by the VDT, EDG and LCG projects should be used to provide compatibility with existing services and procedures.

4.1.3 Resource and task description

Another important condition of interoperability is that all resources that are available on the Grid, as well as all tasks that need to be executed, advertise themselves using a common format or language. In line with the choices made for the currently LCG Grid system, we recommend the use of Condor JDL (Job Description Language)[19] and elements of the GLUE schema[31] for resource and VO configuration and management.

4.2 DESCRIPTION OF ARDA SERVICES

The Open Grid Services Infrastructure offers a convenient model for starting instances of Grid services on demand by means of the Service Factory interface (unlike bare Web Services currently used in AliEn and other projects, where this functionality has to be implemented by the framework in a non-standard way). The Service Factory interface allows for creation of a service port with associated messaging protocol on a per user basis with an appropriately constructed User Interface that reflects roles and capabilities of the authenticated user. For example, VO administrators or production users can have an extended interface, allowing them to do more advanced or higher priority tasks than ordinary users can.

One must assure that there are commonly used Authorisation and Auditing Services and that all communication between components goes via standard channels and agreed interfaces. This allows the exchange of potentially VO-specific components like the Metadata or File Catalogues and allows a VO to replace, on the basis of its needs and preferences, default component with the one provided by experiment, that is adhering to the agreed interface.

This means that File and Metadata Catalogues must appear in the architecture as independent services. Similarly, a Package Manager should be a common service capable of resolving VO specific package dependencies and potentially installing or updating them across Grid sites, rather than of being part of the Computing Element component definition. While in AliEn the state and history of running jobs are accounted for in the File Catalogue, this functionality should be implemented through an additional Job Provenance service with possibly a richer interface that can be used by several other components, like the User Interface or Grid Portals.

As an example, the decomposition of the AliEn model of web services fitting these constrains arrives at a picture of interacting services as shown in Figure 6.

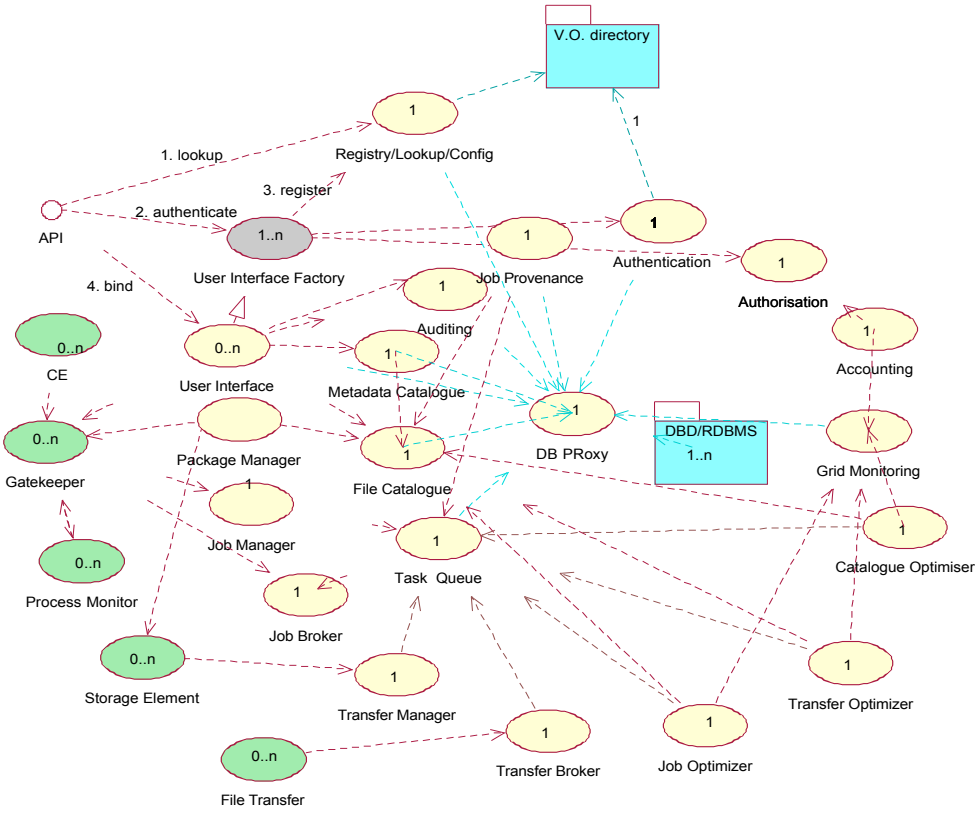

Figure 6: AliEn services expanded to take into account the ARDA requirement to be able to use alternative implementations of key services such as File or Metadata catalogues. This results in a need to expose several software components of AliEn as proper Web Services.

To abstract the implementation details and retain manageable granularity of components, the services are grouped together into logical groups, which correspond to the usual components found in Grid implementations. Taking this approach, we derive the decomposition in the following key ARDA services:

- API and corresponding Grid Access Service Components
- Authentication, Authorisation, Accounting and Auditing Services

- Workload and Data Management Systems
- File and Metadata Catalogue Services
- Information Service
- Grid and Job Monitoring Services
- Storage and Computing Element Services
- Package Manager and Job Provenance Services

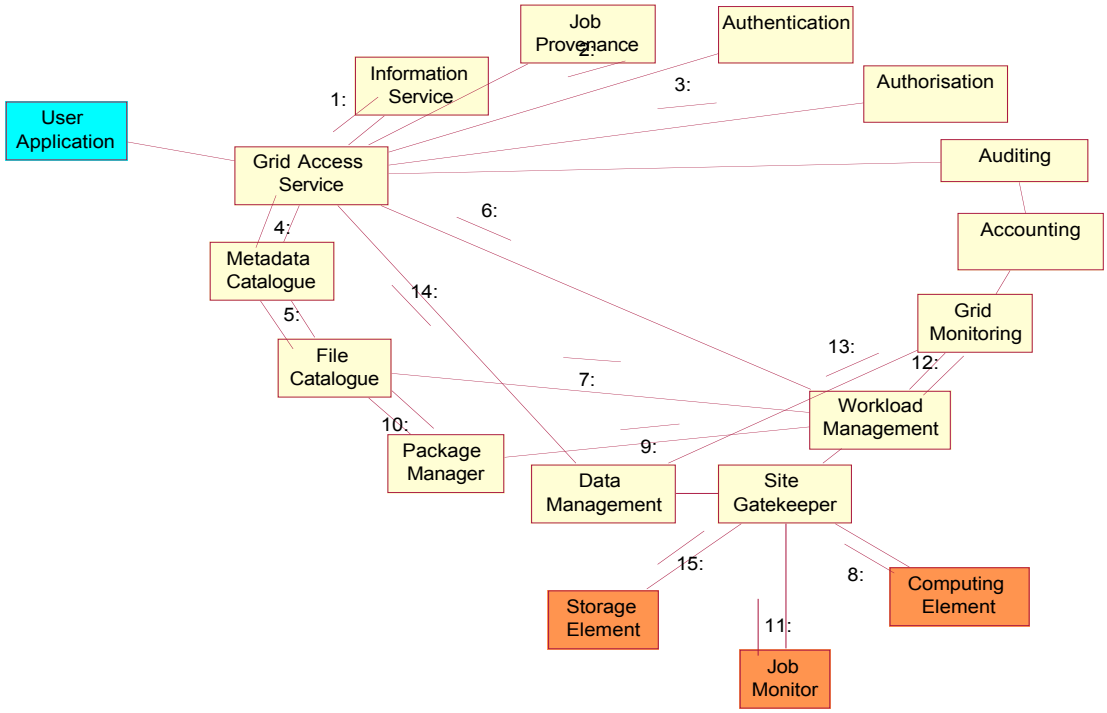

Figure 7: A typical interaction between core ARDA Services and a "user" client, application framework, or another component Grid service

In this picture, the user (or application framework, client tool, component service, etc.) interacts with the service framework by means of an API that has basic interfaces as illustrated in Figure 7.

In the rest of this section we give brief descriptions of the services identified. Some of these services may be in fact complex distributed applications, but their clients should not know anything about these internal details.

4.2.1 API – Accessing ARDA Services

We envision several means by which users and application frameworks will gain access to ARDA services. An ARDA API, shown in Figure 8, 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 files onto the Grid. We consider files available on the Grid to be those stored on one or more Storage Elements and registered in the File Catalogue or replica location service.

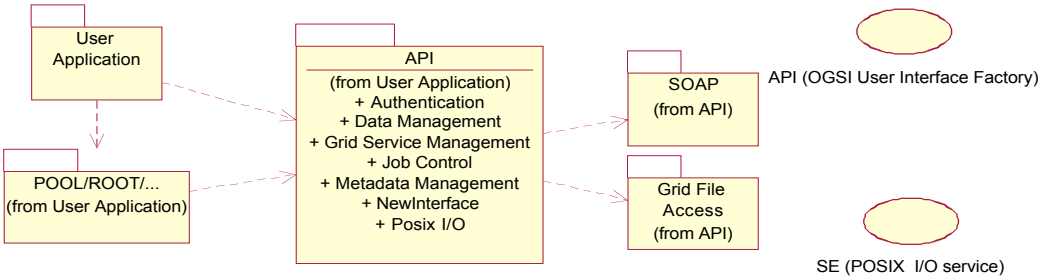

Figure 8: Grid API for user and grid interactions

It is anticipated that developments in service oriented computing, such as workflow composition languages and protocols using web services (such as emerging efforts of BPEL4WS [32] and W3C initiatives in web service coordination, e.g. WSCI [33]) could be exploited to provide a richer set of capabilities for the ARDA GAS/API. Grid Service components can be used to specify processes, data flows, and control mechanisms. The Grid Service components will present a uniform public interface in terms of uniform description of the constituent services, and specification of its input/output messages. Advanced applications referencing core ARDA services could be developed using a Grid Service Component Library and a construction framework for service composition (Figure 9). The application building framework provides a means to describe workflow logic, data flows, and control mechanisms for service instances created from the GAS/API.

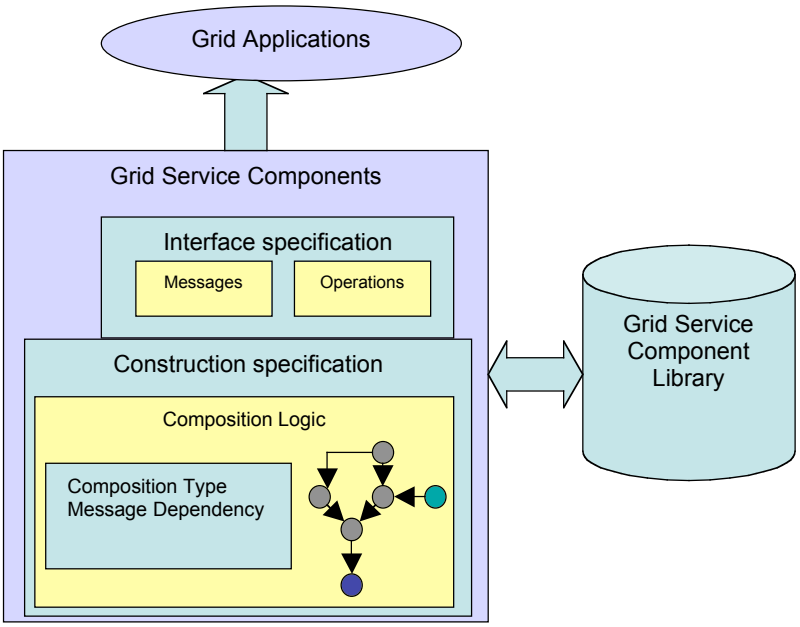

Figure 9: Construction of Grid service coordinators: Ingredients for building Grid applications by composing application and core ARDA services using Grid Service Component libraries, adapted from [34]

At the present time OGSI as such does not specify service behaviours at this level, though discussions in the OGSA community are beginning to address issues of process coordination for Grid service workflows[34]. As technology develops in these areas, ARDA should expose more complex analysis process workflow capabilities using frameworks and composition patterns as indicated here.

4.2.2 Grid Access Service

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 or she 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 10. During its creation the user is authenticated and his or her 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.

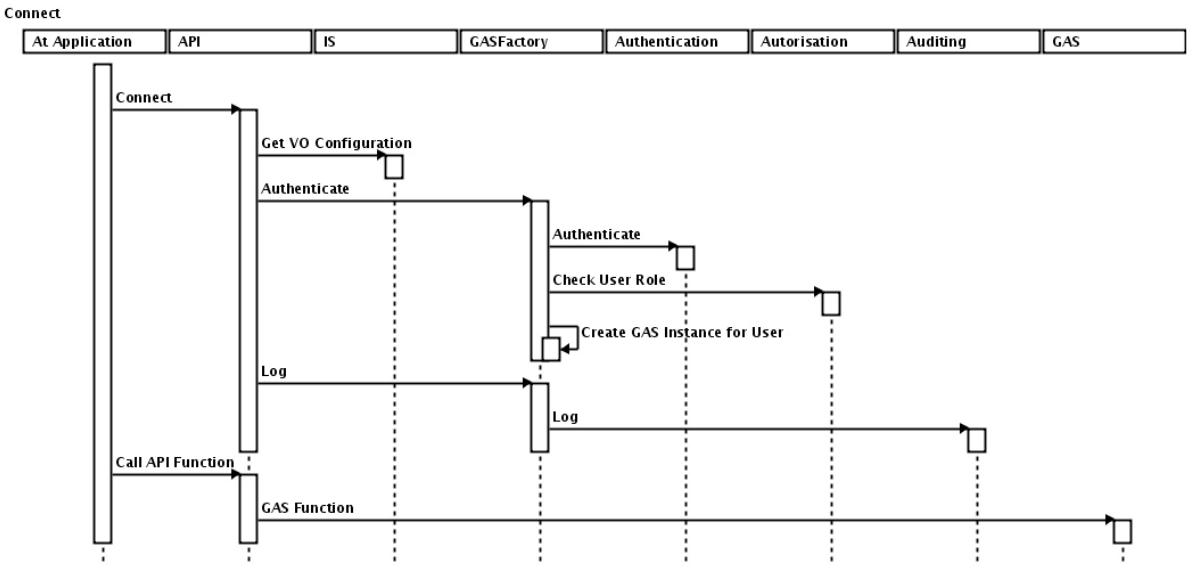

Figure 10: The sequence of interactions between ARDA services while an application initiates a service instance, which provides its connection to the Grid

4.2.3 Information Service

Information Service provides information about the available services and their configuration. It contains mostly static information about the service locations and capabilities. This information is typically used by services to lookup and discover other appropriate services on the Grid. The dynamic status information, e.g. the status of the occupancy of the Grid resources, is available from the dedicated Grid Monitoring. VO-specific information and information about user in a given VO may be part of this service.

4.2.4 Authentication Service

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

4.2.5 Authorisation Service

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

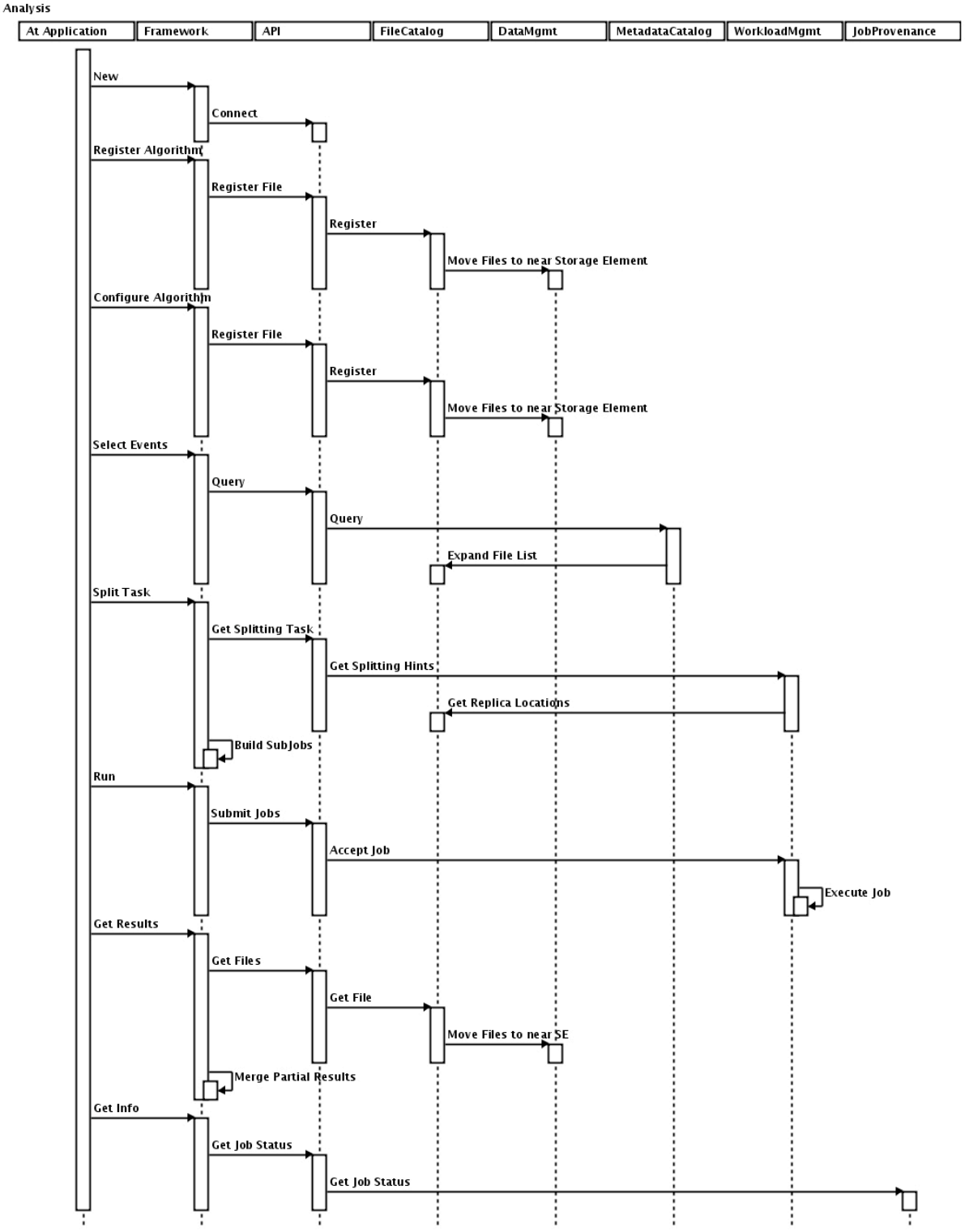

Figure 11: The sequence of interactions betweens ARDA services with an application (experiment analysis framework) executing a typical analysis workflow.

4.2.6 Auditing Service

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

4.2.7 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.

4.2.8 Workload Management Service

The Workload Management Service (WMS) receives the workload instructions from the users (and potentially other services acting on the user's behalf) 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.

4.2.9 Job Provenance Service

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.

4.2.10 File Catalogue Service

The File Catalogue Service keeps association between the logical file names (LFN) and their physical replicas – physical file names (PFN). It contains also minimal metadata information, like file sizes, check sums or file ownership which can serve for the data integrity checks as well as help accomplishing data transfers, caching, etc. The File Catalogue organises its information in a hierarchical structure. The users and groups of users can have private directories with restricted access.

4.2.11 Metadata Catalogue Service

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.

4.2.12 Data Management Service

Data Management is a service to manage scheduled data transfers. These transfers can be a part of planned data replication (mirrors) or triggered by application access to logical files for which physical file does not exist on near Storage Element. Therefore each data transfer can be scheduled and managed very much like an ordinary job. Note that interface and utilities to upload files to and delete from the Grid as well as to get access to them are contained in the API library.

4.2.13 Site Gatekeeper

The Proxy Service is running on the gatekeeper host of a CE. It is used to pass messages to and from the instances of the Job Monitoring Service (see below) that are running on a Worker Node possibly disconnected from the WAN.

4.2.14 Storage Element

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.

4.2.15 Computing Element

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.

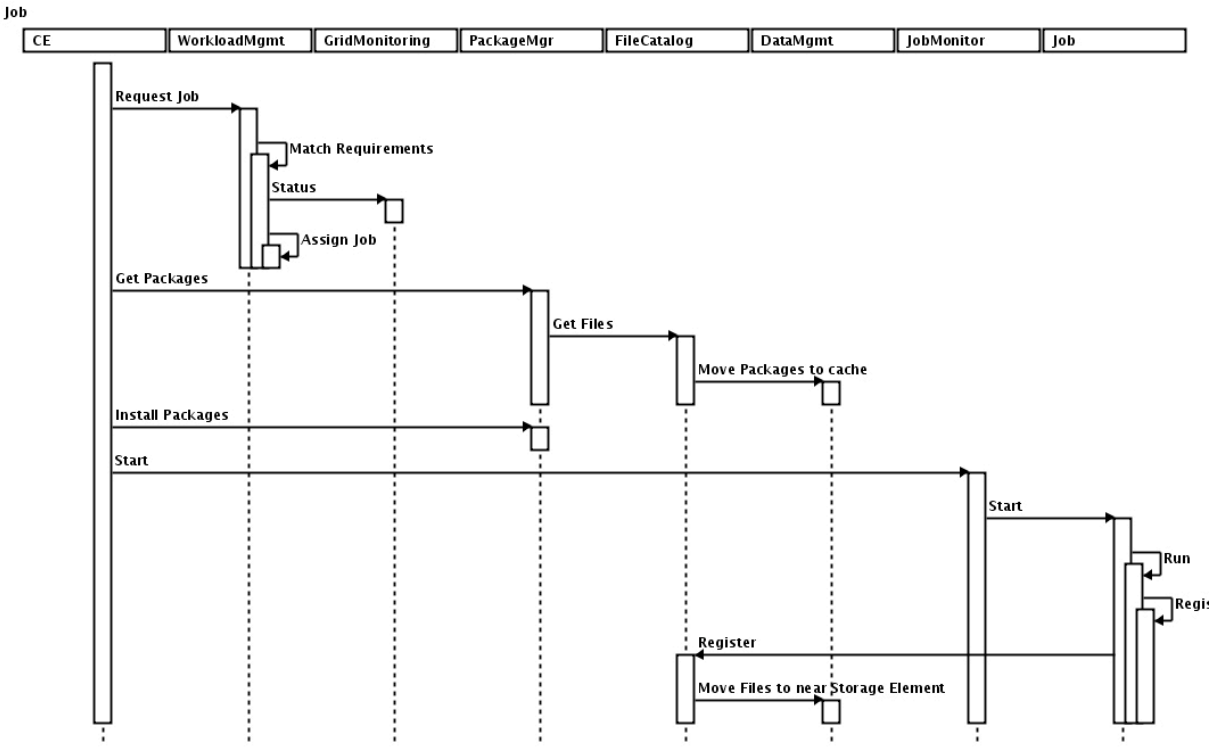

Figure 12: The sequence of interactions between ARDA services illustrating possible job execution model

4.2.16 Job Monitoring Service

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.

4.2.17 Package Manager Service

The Package Manager Service is a specialised database for available software packages of a given VO. 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 CEs.

4.2.18 Grid Monitoring Service

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.

5 THE ARDA PROTOTYPE

The main goal of an ARDA prototype is to provide a more complete blueprint and to develop the specifications for functionality and interfaces of the ARDA services and API.

The ARDA prototype would allow investigating the LHC requirements on distributed analysis further. We recognize the value of real prototyping of services, their functions and interfaces, providing insights how the services implement the required functionality for distributed analysis. Only a prototype will allow to realistically investigate the extent of the possible commonality between the experiments in the API, which is ultimately necessary for the LHC experiments to run their competitive physics research on top of a shared multi-VO environment.

It has been pointed out that there is no "evolutionary" path from the current GT2-based LCG infrastructure, building upon the VDT[6] and EDG[7] components and software stack, to a Grid services architecture based on OGSI compliant services model. The prototype addresses this as it enables to perform real-world OGSI modeling, functionality and performance tests and to address issues how to deploy and run ARDA services along with the existing ones on the LCG-1 resources. There are several areas where the ARDA services would leverage the emerging experience with running a large distributed environment and exploit the R&D in the Grid middleware projects, e.g. in the area of VO management, security infrastructure, set-up of the Computing and Storage Elements etc.

We recommend that the LCG setup a project to develop the prototype, considering these main goals. The schedule and milestones should commensurate with the need to expose the resulting functionality and interfaces early to the community, and be on a timescale of 6 months.

Grid services descriptions and Grid services instances should be defined early in the project. This should include descriptions and specification of core ARDA services, the definition of the external interfaces, that is, the API to experiment frameworks, services and tools, and the interfaces to infrastructure, facilities and fabrics. It is here where experiment developers and other Grid efforts should be engaged from the beginning, and an initial release of the ARDA prototype should be made available to the experiments and the Grid community. The internal interfaces between services should be defined and documented with the end of the prototyping phase of about 6 months. Specific points of interactions with the community should be provided, through early releases, workshops, and actively seeking feedback on API, service interfaces and functionalities.

We recommend that the ARDA prototype project start with a careful definition of the work areas. The constituency of the project and the project lead should be identified quickly, so that a team can be built. The project should develop and present the work plan, schedule and milestones, including a plan for interfacing to and engaging of LHC experiments and the LHC-related Grid community.

We propose a four-prong approach towards these goals:

1. Re-factoring of AliEn and possibly other services into ARDA, with a first release based on OGSI::Lite[8]; consolidation of the API working with the experiments and the LCG-AA; initial release of a fully functional prototype. Subsequently implementation of agreed interfaces, testing and release of the prototype implementation.
2. Modeling of an OGSI-based services infrastructure, performance tests and quality assurance of the prototype implementation
3. Interfacing to LCG-AA software like POOL and ROOT
4. Interfacing to experiment's frameworks, with specific meta-data handlers and experiment specific services

The LCG Application Area, the LCG Grid Technology Area, and possibly the EGEE middleware team, should be involved in the ARDA. We recommend the experiments be involved from the beginning by working on the interface with the frameworks and integrating into the experiment's data and metadata management environments. Other LHC-related projects should be engaged by exposing services and GAS/API definitions early to allow synergistic developments from these projects. It is

important that the appropriate emphasis and effort is being put on documentation, packaging, releases, deployment and support issues.

5.1 EXTENDING ARDA SERVICES

After the prototype phase, we expect a period of scaling-up and re-engineering the OGSI foundation, the database, the information services etc.; deployment and interfaces to site and Grid operations, VO management, information services infrastructure etc.; building higher level services and experiment specific functionality; work on interactive analysis interfaces and new functionalities.

The list of ARDA services discussed in this document is by no means complete. It is likely that a fully functional system will require additional services, for example management of virtual data, handling of experiment specific services tied to persistency and event access models, specific VO policy services or high level optimization and supervision services.

The ARDA blueprint offers the basic functionality required to fulfill the common needs of the LHC experiments, while allowing building upon the basic services described in this document. This approach provides opportunity for well-aligned further developments and the inclusion of new and advanced functionalities.

6 REFERENCES

Acknowledgements

We gratefully acknowledge the contributions from Massimo Lamanna(CERN) to the ARDA RTAG, who participated for a large fraction of the RTAG representing the LCG Grid Technology area.

We also acknowledge and thank for the input and contributions from many, from inside and outside the LHC, in particular those who have given presentations to the RTAG, discussed with us or have given their input in any way. (To Be Completed).

[1] Hepcal-II ref – the RTAG saw and discussed an initial draft version of the Hepcal-II report

[2] AliEn, http://alien.cern.ch,
CHEP2003 Proceedings: http://arXiv.org/abs/cs/0306067,
http://arXiv.org/abs/cs.dc/0306068,http://arXiv.org/abs/cs.dc/0306071, http://arXiv.org/abs/physics/0306103

[3] OGSI, http://www.gridforum.org/ogsi-wg/

[4] GT3, the Globus Toolkit 3, http://www.globus.org

[5] GT2 reference

[6] VDT reference

[7] EDG reference

[8] OGSI::lite reference

[9] EGEE reference

[10] DIAL reference

[11] Clarens, http://clarens.sourceforge.net/, CHEP2003 Proceedings: Clarens Client and Server Applications. Published in eConf C0303241:TUcT005,2003

[12] GANGA reference

[13] DIRAC reference

[14] Initial ARDA presentation to the SC2

[15] Final ARDA presentation to the SC2

[16] PROOF reference

[17] ROOT reference

[18] Web Services reference

[19] CONDOR reference

[] don't know how to get rid of this line...we love MSword

[21] SRB reference

[22] WIRED reference

[23] CAIGEE reference http://pcbunn.cacr.caltech.edu/GAE/GAE.htm.

[24] SPHINX reference

[25] MonALISA, http://monalisa.cacr.caltech.edu/

[26] JAS reference

[27] PPDG CS11 reference

[28] Web Service Definition Language, W3C specification, http://www.w3.org/TR/wsdl

[29] SOAP 1.1 messaging protocol, http://www.w3.org/TR/SOAP/

[30] GSS-API reference

[31] GLUE schema reference

[32] Business Process Execution Language for Web Services (BPEL4WS), http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[33] Web Service Choreography Interface (WSCI), http://www.w3.org/TR/wsci/

[34] “Web Service Componentization”, J. Yang, Communications of the ACM, Vol 46, No. 10 (October 2003).