Grid Interoperability report

Project Number: FP6-2005-IST-026996

Deliverable id: D 3.5

Deliverable name: Grid Interoperability report

Submission Date: 19/08/2010

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	ACGT
Project Full Name:	Advancing Clinico-Genomic Clinical Trials on Cancer: Open Grid Services for improving Medical Knowledge Discovery
Document id:	D3.5
Document name:	Grid Interoperability report
Document type (PU, INT, RE)	PU
Version:	1.0
Submission date:	19.08.2010
Editor: Organisation: Email:	Juliusz Pukacki PSNC pukacki@man.poznan.pl

Document type PU = public, INT = internal, RE = restricted

ABSTRACT

Deliverable presents the ideas concerning interoperability of different components of ACGT architecture in a context of integrating some tools or services with other testbeds or exploiting external infrastructures for computation or storage.

KEYWORD LIST: System Architecture, Service Oriented Architectures, Architecture Design Patterns, Grid, Distributed Computer Systems, Globus, Gridge Toolkit

MODIFICATION CONTROL			
Version	Date	Status	Author
0.1	27.06.2010	Draft	J. Pukacki
0.2	20.07.2010	Draft	J. Pukacki
0.3	15.08.2010	Draft	J. Pukacki
1.0	30.08.2010	Final	J. Pukacki

List of Contributors

- Stelios Sfakianakis, FORTH
- Dennis Wegener, FhG
- Pawel Spychala, PSNC

Contents

Executive Summary

ACGT is an Integrated Project (IP) funded in the 6th Framework Program of the European Commission under the Action Line “Integrated biomedical information for better health”. The high level objective of the Action Line is the development of methods and systems for improved medical knowledge discovery and understanding through integration of biomedical information (e.g. using modelling, visualization, data mining and grid technologies). Biomedical data and information to be considered include not only clinical information relating to tissues, organs or personal health-related information but also information at the level of molecules and cells, such as that acquired from genomics and proteomics research.

ACGT focuses on the domain of Cancer research, and its ultimate objective is the design, development and validation of an integrated Grid enabled technological platform in support of post-genomic, multi-centric Clinical Trials on Cancer. The driving motivation behind the project is our committed belief that the breadth and depth of information already available in the research community at large, present an enormous opportunity for improving our ability to reduce mortality from cancer, improve therapies and meet the demanding individualization of care needs.

1. Introduction

The term interoperability refers to the ability of the systems to work (cooperate) with the other systems. That kind of system property is important in the context of integration with already existing solution and also for the system flexibility. Flexibility in that context means ability to replace components of the system in the future.

The most common interpretation of interoperability in the IT systems is ability of one system to exchange the data with the other but without understanding its meaning - it is called syntactic interoperability. Much more challenging is provision of automatic interpretation of exchanged data - in that case two systems can work as one from the point of view of the end user (semantic interoperability).

Interoperability issues should be considered on the level of system design and carefully bound into implementation phase. The most important factor that increases the syntactic interoperability is a usage of standards in the process of system development.

Open standards is a valuable initiative that helps to prevent compatibility between different systems, and even implies syntactic interoperability.

Open Standards rely on a broadly consultative and inclusive group including representatives from vendors, academicians and others holding a stake in the development. That discusses and debates the technical and economic merits, demerits and feasibility of a proposed common protocol. After the doubts and reservations of all members are addressed, the resulting common document is endorsed as a common standard. This document is subsequently released to the public, and henceforth becomes an open standard. It is usually published and is available freely or at a nominal cost to any and all comers, with no further encumbrances. Various vendors and individuals (even those who were not part of the original group) can use the standards document to make products that implement the common protocol defined in the standard, and are thus interoperable by design, with no specific liability or advantage for any customer for choosing one product over another on the basis of standardised features. The vendors' products compete on the quality of their implementation, user interface, ease of use, performance, price, and a host of other factors, while keeping the customers data intact and transferable even if he chooses to switch to another competing product for business reasons.

Semantic interoperability gained by using standard is just a first though vary important step. The next level of interoperability can only be achieved on the level of system development.

It requires more implementation effort to translate semantics of one systems to the terms of the other. It is obvious that systems are not one hundred percent compliant thus it is not required to prevent full compatibility. The common models are based on using some functionality of the other system via provided interfaces, and proper interpretation of returned results.

In the next chapter the analysis concerning ACGT infrastructure in the context of interoperability are discussed. It is mostly focused on the lower levels of ACGT architecture that provides access to computation and data storage resources.

2 ACGT architecture

2.1 Key concepts

There are two the most important concepts behind ACGT architecture design. The first one is layering and the second one is Service Oriented Architecture (SOA) concept.

They were used together in a process of design and development of the ACGT infrastructure components. So in some cases final architecture is a kind of compromise between those two ideas.

Layering is one of the most popular architectural design pattern. It structures software, so it can be decomposed into groups of components such that each group of components is at a particular level of abstraction. It also helps to split responsibilities of the different groups of components in the system.

At first layered architecture were used to divide application presentation features from application logic and database access (three-tier model). The highest level layer in that model is used as an interface for the end user that translates tasks and results to something the user can understand.

Logic layer coordinates the application, process commands, makes logical decisions. It also moves and processes data between two surrounding layers. Within Data Layer all the information required by the application is stored and retrieved from storage databases or filesystem.

Such a classical approach as described above can be used as a reference point for more complex applications and systems. It is possible to extend basic model by spiting particular layers to additional ones depending on the characteristics of the concrete problem. The actual meaning of the layers also can be result of modelled environment.

For the ACGT infrastructure the layers denotes group of components performing some operations more ore less specific to the bio-medical background of the project. The upper layers (closer to the end user) are strongly connected to the terms and ideas from the biology and medicine areas. They are using specific language, try to benefit from ontology developed within the project, and staying distant from the technical infrastructure behind - computers, networks, storages, etc.

The lower layers were design to be universal in the context of operations performed. Their main role is to provide an access to physical resources in the most convenient way. Components of that layers are not aware of problem specific vocabulary, they are not using ontology descriptions, and can be easily replaced with the different ones with similar functionality if necessary.

There are some principles behind the layers design. The important one is to implement the services in the same layer using the same or similar technology. So the communication between components within the layer is based on the same mechanisms. It makes development process much easier and makes possible to introduce some guidelines for software developers. This rule is especially important for the layers that have a lot newly developed components. I the case of reusing the existing infrastructure it is not as much significant.

The other rule connected with layering is clear layer separation in the terms of communication between layers. It means that component of one layer can only communicate to the components of the layer directly below. It makes architecture much more clear and better structured. Of course some exception from this rule are accepted. For instance some logical functionality can be used by the components of all layers, and it is pointless to replicate the same element throughout all layers.

Nowadays, one of the most important trend in system design is Service Oriented Architecture (SOA). It defines software infrastructure as a loosely coupled software services to support the requirements of the business processes and software users. In an SOA environment, resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation. A service-oriented architecture is not tied to a specific technology. It may be implemented using a wide range of interoperability standards but the most important standard in the context of SOA is Web Services. The key issue is that all services that building some environment are independent from each other, they just publish their interfaces and that is enough for any other entity in the system to invoke methods means 'use' the service.

At first sight it is maybe difficult to imagine the how the SOA concept coexists with layering paradigm. The explanation is that they operates on different levels of abstraction. Layering concerns functionality of services and the distance from the end user and physical resources, on the other hand SOA describes the communication pattern.

The two concepts together constitutes the logical framework and provides clear guidelines for service developers. So for the newly developed service at first it the location within the layers must be defined. The decision about it is based on the role of the service, if it will be used directly by the end user, or if it provides access to physical resources. When it is decided development of the service should follow SOA rules: it should be independent entity accessed via network through the well defined interface. It should work as a standalone component or provide functionality cooperating with the other services.

2.2 Design view

The role of the initial architecture was to deliver basic framework for infrastructure developers. It was supposed to help all technical partners in ACGT consortium in development of their software components. At the beginning of the project there was no detailed specification of the required services to be created, so the initial architecture was intended to be a framework - a design view of the future architecture.

The picture below presents that general design view as a starting point for development of ACGT infrastructure.

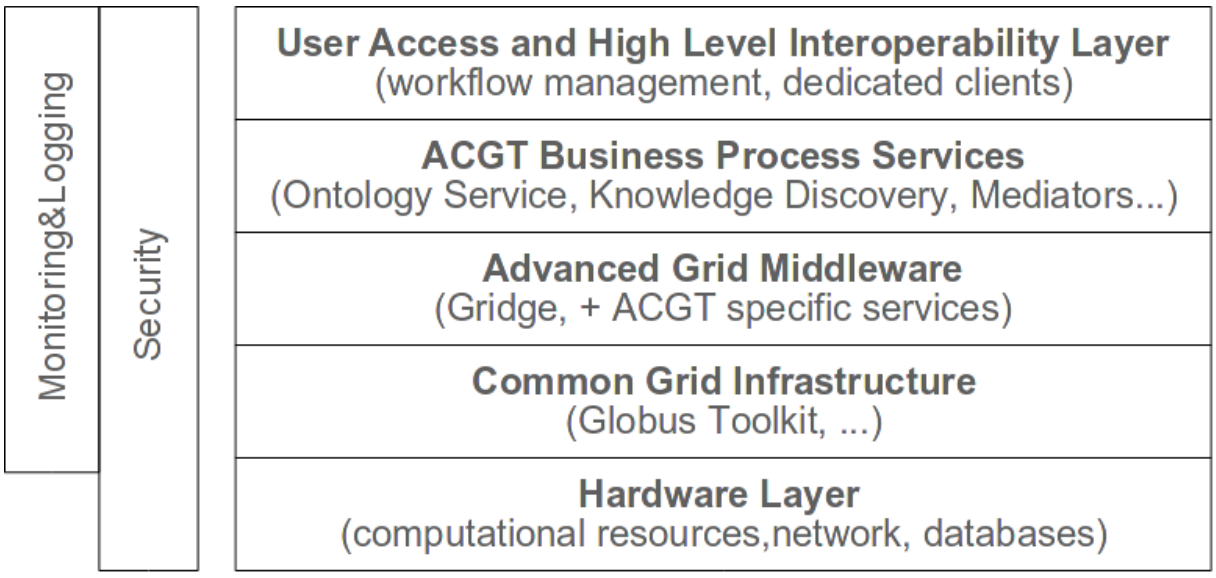


Fig.1. ACGT layered architecture.

There are five horizontal layers presented on the picture. The lower ones are located closer to physical resources. The mechanisms used for distributed access to resources is grid technology. The layers at the top are responsible for providing ACGT specific solutions for bioinformaticians and clinicians.

There are also two vertical layers. The first one is logging infrastructure and is used by the services regardless the location within architecture. It s very important to have ability to track one the activity that in many cases involves different services from different layers.

The other vertical layer is security that constitutes common infrastructure for all components in the infrastructure. It is very important to keep consistent security policies throughout the infrastructure. And also to be able to dynamically manipulate the policies in the context of virtual organisation management.

2.3 Layers description

As it is marked on the picture with green colour the grid technologies are present on the three lower layers. The lowest one are the hardware resources where the computation is done and where physical data is stored. The role of the next layer above the hardware resources is to provide unified, remote access to physical resources. It provides basic functionality required for remote computing and data access:

- job execution and control
- basic authentication and authorization
- file transfer
- databases access
- hardware monitoring
- information about state of resources (static and dynamic metrics)

The last Grid layer is Advanced Middleware Layer. It is responsible for providing more advanced mechanisms in the Grid environment. Services from this layer can be described as "collective" because they operate on set of lower level services, to realize more advanced actions - e.g. metascheduling service that submits jobs to different local queuing systems using Common Grid Infrastructure remote interfaces.

Functionality provided by this layer can be gathered in a following main points:

- resource management - metascheduling
- data management (database access and file storing and transferring)
- services authorization
- grid monitoring

As it can be seen on architecture picture the Grid layers are separated from the rest of the system that is build with services that provides specific ACGT content.

The advantage of it is clearly visible: based on that Grid platform it is possible to build many different environments for different fields, not only biomedicine.

Grid layers are supposed to provide standard and secure way for accessing hardware resources of the Grid environment.

ACGT business process layer consists of components which are not aware of physical resources and Grid environment. Grid is used as a whole, to perform more abstract actions,

and to get information required by end user. Services of this layer are using abstract description of the world defined as some ontology.

There are also services that are able to translate high level description to language understandable by lower layer (grid) . It provides higher level integration of different resources and data and makes them more similar to real word objects.

It is possible do define following functionality for Business Processes Services:

- ontology description provisioning
- execution and control of workflows
- translation of abstract object to resources names
- translation of abstract actions to grid calls
- virtual organization management
- knowledge discovery
- biomedical algorithms

Services of this layer are specific for ACGT environment, but in some cases it is possible to reuse existing ones. For instance it is very important to include analytical services that are already there, into ACGT infrastructure. The level of integration depends on security requirements. If there are no restrictions in using the service it is possible to skip the integration on the level of authentication and authorization. But there are more problems with introducing security policies for the third party components. They require ACGT compliant wrappers responsible for taking care of access rights management that is consistent with the whole security policy of ACGT. More detailed description of the security policies of ACGT can be fount in chapter 5.

The User Access Layer contains all application and tools that provide access to the ACGT Environment for end user. There can be a wide variety of software, developed using different technology:

- portals
- standalone applications
- clients dedicated to specific operation
- workflows editors
- visualization tools

Client applications are in the most cases tightly connected to functionality provided by underlying layer. They are also going to be used by different but specific groups of users. Thus it would be difficult to use general tool but there will be need to create dedicated application. These two arguments are showing that User Access Layer will be mostly developed by ACGT.

3. Interoperability in ACGT infrastructure.

3.1. Introduction

ACGT infrastructure architecture model presented in a previous chapters was designed taking into account interoperability requirements.

The layers of the architecture that gathers the components on the same abstraction level were introduced to provide ability to exchange components in a flexible way or to add new components with new or alternative functionality.

It does not mean that it is possible to use replaced or added component without any development effort. But with precisely defined layers it is much easier to detect what kind of work is required to exploit some new functionality. In case of replaced components most of the development is focused on interfaces between layers.

The second paradigm playing important role in the ACGT architecture design - SOA - makes adding new services very easy from the architecture point of view. The reason of that is that there are no stiff rules and exact paths of communication among components. If there is a new end user scenario that requires a new functionality on new services can appear on different layers fulfilling user requirements.

As it was described in previous chapter, there were some common requirements for the technology of implementation of services for different layers. The most important one in Web Services technology based on SOAP messages with some additional security components, namely GSI (Grid Security Infrastructure).

Because of the specificity of the security problem it has to be present on each layer of the architecture and has to be consistent throughout the infrastructure. That kind of special role of the GSI in the ACGT infrastructure could be a source of problems with interoperability but on the other hand GSI is the most important standard in a grid world that makes it the most reasonable choice for the security framework of the whole infrastructure.

3.2. Standards in ACGT

The most important standard for the ACGT infrastructure implementation are:

3.2.1. WSDL

Web Services Description Language - XML based language for describing interfaces of Web services. The WSDL defines services as collections of network endpoints, or ports. The WSDL specification provides an XML format for documents for this purpose. The abstract definition of ports and messages is separated from their concrete use or instance, allowing the reuse of these definitions. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Messages are abstract descriptions of the data being exchanged, and port types are abstract collections of supported operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding, where the messages and operations are then bound to a concrete network protocol and message format. In this way, WSDL describes the public interface to the web service.

WSDL is the most important standard used for Web services implementation. In ACGT project all services implemented are described using WSDL.

3.2.2. GSI

Grid Security Infrastructure is a specification for secret, tamper-proof, delegatable communication between software in a grid computing environment. Secure, authenticatable communication is enabled using asymmetric encryption.

A central concept in GSI authentication is the certificate. Every user and service on the Grid is identified via a certificate, which contains information vital to identifying and authenticating the user or service. A GSI certificate includes four primary pieces of information:

- subject name, which identifies the person or object that the certificate represents.

- the public key belonging to the subject.

- the identity of a Certificate Authority (CA) that has signed the certificate to certify that the public key and the identity both belong to the subject.

- the digital signature of the named CA.

The GSI uses the Secure Sockets Layer (SSL) for its mutual authentication protocol. By default, the GSI does not establish confidential (encrypted) communication between parties. Once mutual authentication is performed, the GSI gets out of the way so that communication can occur without the overhead of constant encryption and decryption.

The GSI provides a delegation capability: an extension of the standard SSL protocol which reduces the number of times the user must enter his pass phrase. If a Grid computation requires that several Grid resources be used (each requiring mutual authentication), or if there is a need to have agents (local or remote) requesting services on behalf of a user, the need to re-enter the user's pass phrase can be avoided by creating a proxy.

In ACGT project GSI was introduced as a common security infrastructure not only for the services of Grid layer, but for all services in ACGT environment.

3.2.3. JSDL

Job Submission Description Language is an extensible XML specification from the Global Grid Forum for the description of simple tasks to non-interactive computer execution systems. Currently at version 1.0 (released November 7, 2005), the specification focuses on the description of computational task submissions to traditional high-performance computer systems like batch schedulers.

JSDL describes the submission aspects of a job, and does not attempt to describe the state of running or historic jobs. Instead, JSDL includes descriptions of:

- Job name, description
- Resource requirements that computers must have to be eligible for scheduling, such as total RAM available, total swap available, CPU clock speed, number of CPUs, Operating System, etc.
- Execution limits, such as the maximum amount of CPU time, wallclock time, or memory that can be consumed.
- File staging, or the transferring of files before or after execution.
- Command to execute, including its command-line arguments, environment variables to define, stdin/stdout/stderr redirection, etc.

In ACGT project JSDL is used by resource management system for Grid (GRMS) that is used for submission of computational jobs to the Grid.

3.3.4. SPARQL

Within ACGT we use SPARQL [1] as a query language. It is supported by the data access services that wrap the various data sources. The semantic mediator uses SPARQL as well, not only in constructing the queries that are sent to the data access service, but also for receiving queries (expressed in the ACGT Master Ontology).

The choice for SPARQL as the common query language was made after first identifying the main requirements, and subsequently evaluating various candidate query languages, including SQL and XQuery. SPARQL was judged the most suitable, as it is based on a general graph-based model, and because it is less complex than the other query languages. SPARQL has an intermediate level of expressiveness, which was also deemed appropriate given the range of data sources that need to be integrated into the ACGT architecture. A more detailed justification for the use of SPARQL can be found in D5.2 [2], which also gives examples of how it is used.

For returning the results of SPARQL queries we are using the SPARQL Query Results XML Format [3], which is as the name suggests a way to return the query results using XML. Given the choice to use Web Services and SPARQL as the common query language, this is therefore a logical choice. D5.2 includes an example that shows how results are formatted using the SPARQL Query Results XML Format.

3.3.5. RDFS

RDFS is a knowledge representation language, which final W3C recommendation was released in February 2004. It is designed to describe the schema of an RDF repository. RDF is a general-purpose language that allows describing resources. Its XML syntax is designed to permit a high compatibility among different applications, even when covering heterogeneous domains. RDF has been developed by the RDF Core Working Group as part of the W3C Semantic Web Activity. In RDF, resources are divided in classes. A class defines a group of elements sharing the same properties. Each class may have a series of attributes called *properties*. These properties can connect a class with another class or with a basic type. Classes are arranged hierarchically, so that subclasses of a given class inherit the properties of that class.

The OWL language for describing ontologies is built on top of RDFS. It inherits its syntax, adding some extra capability for defining constraints and different types of properties. This language was chosen in ACGT to specify the Master Ontology. OWL is currently the most used ontology language.

RDFS defines a series of XML tags that allow specifying the classes and properties that an RDF document can make use of. Next, a brief description of the most important tags is presented:

1. rdfs:Class: allows defining RDF classes. The name of the class is specified though attributes inside that tag
2. rdfs:subClassOf: this tag is placed inside a class declaration, and allows defining the class hierarchy by indicating the superclass of the class being defined
3. rdf:Property: allows defining an RDF Property. Detailed information, such as domain and range, is specified by means of other tags
4. rdfs:domain: declares the domain of a property (usually a class)

5. rdfs:range: declares the range of a property

In ACGT, RDFS is used to describe the schemas of the databases to be integrated in the Semantic Mediator. In some way, it is also utilized to describe the ACGT Master Ontology—in fact, OWL-DL is the language in which the MO is written, however this is not but an extension of the RDFS language, as it was explained before.

RDFS is also the language for exposing the Global Schema—a schema representing the set of queries accepted by the mediator. This schema is generated automatically from the mappings of the integrated databases with the MO. To create it, two steps are carried out sequentially: i) the RDFS information of the MO is extracted, generating an RDFS document —unnecessary information for the global schema, such as some types of OWL-DL specific restrictions, is eliminated— and ii) intersection of this information with the set of mappings with databases is generated, providing the final Global Schema to be used by the Semantic Mediator.

3.2.6. BPEL

In the context of WP9 'The Integrated ACGT Environment' the Web Services Business Process Execution Language (WS-BPEL) is used. Standardized by the OASIS organization, WS-BPEL is a language for specifying business process behaviour based on Web Services. It defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces. The processes described in WS-BPEL can be one of two kinds: Executable and Abstract processes. Executable business processes model actual behaviour of a participant in a business interaction. Abstract business processes are partially specified processes that are not intended to be executed and they may be used to hide some of the required concrete operational details.

In the ACGT platform WS-BPEL is the technology used to support the high level integration of the ACGT services and tools into complex scientific workflows for the implementation, testing and validation of user scenarios. The workflows defined in WS-BPEL are deployed as executable processes and they can be subsequently used as atomic services to construct even more complex and higher level scenarios.

Much of the effort in ACGT that is related to WS-BPEL has been focused on making the tools that implement this standard more compliant with the rest of the ACGT architecture. Notable work areas are the invocation of Grid Services and the integration of Grid Security Infrastructure.

3.3. ACGT grid infrastructure interoperability

The way of providing interoperability differs depending on abstraction layer of architecture. General assumption is that interoperability on lower layers should be transparent from the end user of the infrastructure. It means that it should be possible to exploit different grid infrastructures without bothering ACGT clients, and providing them new computational power and storage space in a transparent way. This objective could be achieved by two essential pillars of architecture design:

- separation of grid infrastructure from domain specific part of ACGT services

- splitting grid infrastructure into two layers: basic grid services and advanced grid middleware

The first issue influences the ability for transparent exploitation of grid resources. It helps to treat the grid as a black box used via well defined interfaces to submit computation and store data. It is not accessed directly by end users (clinicians, bioinformaticians, etc.) but by domain level services like GridR, workflow management system or Oncosimulator service.

The above-mentioned services are also not touching grid infrastructure directly but through Advance Grid Middleware Layer. This layer is responsible for providing platform of interoperability for the different existing grid solutions. It consists of collective services responsible for resource discovery, metascheduling, data storage, grid monitoring etc.

Below layer of advanced grid services there is Common Grid Infrastructure layer that constitutes remotely accessed interface to physical resources (computational nodes, storage elements, etc.). In a current implementation it is built with services of Globus Toolkit - very popular software for building grid environments. Even the decision of using Globus as a low level grid platform was taken having in mind interoperability issues. The most important features of it in this context are:

- it is very popular in a grid world
- it is based on well-known and commonly used standards
- design and implementation principles of Globus are very generic that makes it easy to replace
- it can be deployed on many different system and can interface to variety of local resource management (queueing) systems: PBS, LSF, SGE, Condor

Globus Services provide basic interfaces for remote job submission, controlling of a job execution (job status changes), execution failure detection and resubmission, information about available resources. In addition to that functionality the important role plays GridFTP - GSI enabled implementation of standard FTP service. GridFTP is used for preparing environment for application execution, transferring input and output data.

To replace Globus infrastructure or to add resources governed by other grid system this basic set of functionalities should be provided:

- remote execution of application
- job control: status changes notification, cancelling, suspending, resuming the execution
- remote access to user account to prepare execution environment
- transferring input data to user account and output data from user space on computational node to some external location

The only hard constraint came from security requirements of ACGT architecture: new system should be compliant with GSI security rules. It would be possible to avoid this only by providing some additional, trusted, security component that would be able to translate security policy from GSI world to some other.

The incorporation of new grid system into ACGT infrastructure is possible to implement on the level of Advanced Grid Services. One of the most important component of this layer is GRMS - grid metascheduler. It is responsible for discovering the resources in the grid and choosing one for execution of application. There are some features that makes GRMS interoperable:

- independent resource model - internal resources description is a very generic one and is not based on any particular, existing implementation;

- independent submission language with an option of using standard JSDL job description
- pluggable architecture - GRMS is design and implemented the way that allows to add support for the different external services as a plugin chosen dynamically or based on configuration.

Support for the different grid environments on the level of metascheduling can be add by in a two general ways:

1) Configuring set of plugins inside one instance of GRMS

One GRMS is able to spit computational jobs between different grid environment based on plugin configuration. There is an assumption that available resources are described in one information service.

2) Setting separate instances of GMRS for different environments.

The assumption of this configuration is to have separate GRMS taking care about submitting jobs to different environments.

With this solution there is an option to set special instance of GRMS on the top of the basic ones that is playing a role of meta-scheduler for other meta-schedulers and create hierarchy of resource management. This option would require to implement a plugin for GRMS and some meta information system providing the information about specific of resources laying below each lower level GRMSes.

There are a project that are trying to solve interoperability issues on a different layers. HPC Europa and HPC Europa 2 projects are the examples where the integration of different grid environment is implemented on the portal level.

3.4. Resource management interoperability

The following resource management system and environments can be used by GRMS for job submission. Some of the systems are already fully integrated (SMOA, GRIA) and some of them requires new or updated plugin to be implementaed.

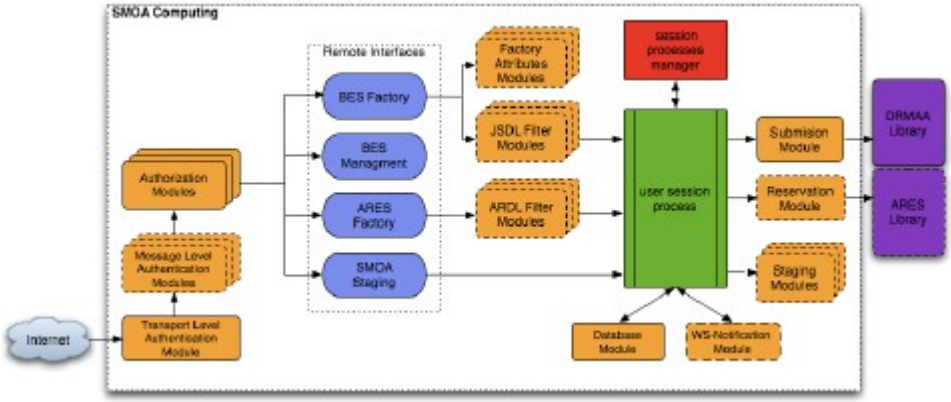
3.4.1. SMOA Computing

In the mid 2005 the OpenDSP (Open DRMAA Service Provider) project was launched at Poznan Supercomputing and Networking Center (PSNC). The main goal of the project was to develop a service giving consistent, remote, multiuser access to various DRM systems using standardized DRMAA interface in the layer between the DRM system and the service. Leverage of the DRMAA [6] C binding and usage of the C language in the core modules was expected to result in high performance of the service. Although the OpenDSP service could accept job described in the JSDL, the remote interface of the service was selfdesigned,and thus it is not a part of any standard. After more than two years, and four successive releases of the OpenDSP, a decision was taken to update the remote interface, and exploit another OGF standard: the OGSA Basic Execution Service (profiled by the HPC Basic Profile specification). With the new interface also the other aspects of the service have changed:

- refined architecture,
- privilege separation instead of setuid binaries,
- JSDL used as the sole format for internal job representation,

- support for modules written in Python,
- more extensions points added,
- exposed Advance Reservation capability of underlying DRMS,
- file staging support as the part of job life cycle,
- separate interface for lightweight, direct file staging (via SOAP attachments).

Also a new name was given to the service: SMOA Computing, as the DSP acronym was quite often misunderstood with Digital Signal Processing.



The above diagram depicts the overall SMOA Computing Architecture. The service interface is composed of 4 Web Service ports:

- BES-Factory - interface for job creation, monitoring and management,
- BES-Management - interface for managing the service,
- ARES Factory - interface for advance reservation creation and management - a SMOA Computing extension (described in section Advance Reservation Interface),
- SMOA Staging - interface for direct (client-service) file transfer via SOAP attachments - a SMOA Computing extension.

The SMOA Computing service was successfully tested with the following DRM systems:

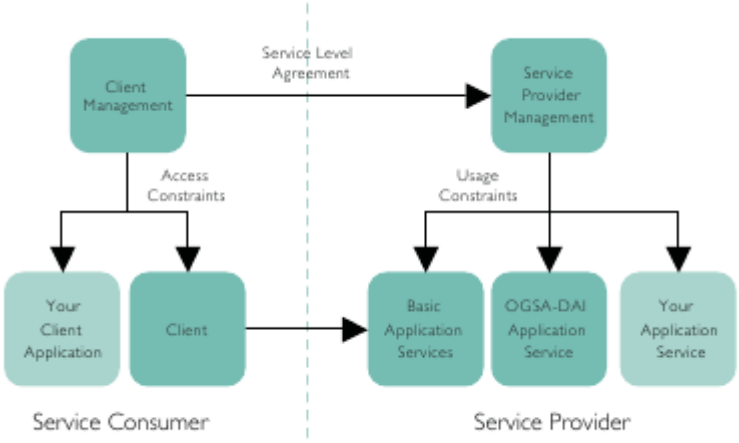
- Sun Grid Engine,
- Platform LSF,
- Torque,
- PBS Pro,
- Condor,
- Apple XGrid.

3.4.2. GRIA

GRIA is a service-oriented infrastructure designed to support B2B collaborations through service provision across organisational boundaries in a secure, interoperable and flexible manner. GRIA uses Web Service protocols based on key interoperability specifications. GRIA is available free and open source (most of the software is LGPL)

GRIA makes use of business models, processes and semantics to allow service providers and users to discover each other and negotiate terms for access to high-value IT assets. By focusing on business processes and the associated semantics, GRIA enables users to provision for their computational needs more cost effectively, and develop new business models for their services.

GRIA provides several software packages and toolkits each created to address the business needs of service consumers and providers. The packages are designed using a service-oriented architecture to support flexible composition scenarios and extensibility.



Basic Application Services allow an organisation with cluster computing facilities to provide data storage and processing (using applications installed on the cluster)

OGSA-DAI Application Services allow an organisation to publish structured data resources (XML, relational, bespoke) for subscription

Client allows user access to GRIA management and application services via desktop applications, includes a client API toolkit for integration of your client applications

Client Management provides optional support for organisation-level management of service users with centralised control and monitoring of service procurement and usage Service Developers Toolkit allows integration of your application services with GRIA's security and management capabilities

Service Provider Management provides optional support for SLAs based service management and billing based on a simple management protocol for both GRIA and 3rd party application services

GRMS Gria plugin adds ability to GRMS to submit jobs on Gria 5.3 Basic Application Service which can be installed on many different host machines. - To authenticate user needs to have proxy credential. User credential is used to set a proper ownership of the job. If a user submits job he becomes owner of the job and output data of the job. - Gria middleware uses JSDL as a standard of the job description. Not all elements of JSDL are supported by Gria but GRMS Gria plugin selects only necessary ones. - To store data Gria middleware uses Data Stagers which are containers for physical files. Each job can has one or more input and output Data Stagers. JSDL that describes a job need to has elements "source" and "target" defined. It is needed to distinguish input and output data files which goes to proper Data Stagers. - GRMS Gria plugin returns REST urls for output Data Stagers after the job is finished. These urls can be used afterwards to download output data file or files. - GRMS Gria plugin has ability to monitor status of the job, User can see if the job is still active or if it is finished or failed. - Jobs can be cancelled by GRMS Gria plugin. In progress - handling

Gria Information Service which gives ability to pick a best available Gria 5.3 Basic Application Service host resource to submit job, in example - on the basis of free starage space.

3.4.3. gLite

gLite is the middleware stack for grid computing used by the CERN LHC experiments and a very large variety of scientific domains. Born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centers as part of the EGEE Project, gLite provides a complete set of services for building a production grid infrastructure. gLite provides a framework for building grid applications tapping into the power of distributed computing and storage resources across the Internet.

The gLite user community is grouped into Virtual Organisations (VOs). A user must join a VO supported by the infrastructure running gLite to be authenticated and authorized to using grid resources.

The Grid Security Infrastructure (GSI) in WLCG/EGEE enables secure authentication and communication over an open network. GSI is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol, with extensions for single sign-on and delegation.

In order to authenticate himself, a user needs to have a digital X.509 certificate issued by a Certification Authority (CA) trusted by the infrastructure running the middleware.

The authorisation of a user on a specific Grid resource can be done in two different ways. The first is simpler, and relies on the grid-mapfile mechanism. The second way relies on the Virtual Organisation Membership Service (VOMS) and the LCAS/LCMAPS mechanism, which allow for a more detailed definition of user privileges.

Components of the system:

- Computing Element
  A Computing Element (CE), in Grid terminology, is some set of computing resources localized at a site (i.e. a cluster, a computing farm). A CE includes a Grid Gate (GG)1, which acts as a generic interface to the cluster; a Local Resource Management System (LRMS) (sometimes called batch system), and the cluster itself, a collection of Worker Nodes (WNs), the nodes where the jobs are run.
- Storage Element
  A Storage Element (SE) provides uniform access to data storage resources. The Storage Element may control simple disk servers, large disk arrays or tape-based Mass Storage Systems (MSS). Most WLCG/EGEE sites provide at least one SE. Storage Elements can support different data access protocols and interfaces. Simply speaking, GSIFTP (a GSI-secure FTP) is the protocol for whole-file transfers, while local and remote file access is performed using RFIO or gsidcap.
- Information Service
  The Information Service (IS) provides information about the WLCG/EGEE Grid resources and their status. This information is essential for the operation of the whole Grid, as it is via the IS that resources are discovered. The published information is also used for monitoring and accounting purposes.
- Workload Management System
  The purpose of the Workload Management System (WMS)[5] is to accept user jobs, to assign them to the most appropriate Computing Element, to record their status and retrieve their output. The Resource Broker (RB) is the machine where the WMS services run. Jobs to be submitted are described using the Job Description Language (JDL), which specifies, for example, which executable to run and its parameters, files

to be moved to and from the Worker Node on which the job is run, input Grid files needed, and any requirements on the CE and the Worker Node.

3.4.4. UNICORE

UNICORE (UNiform Interface to COmputing REsources) is a Grid computing technology that provides seamless, secure, and intuitive access to distributed Grid resources such as supercomputers or cluster systems and information stored in databases. UNICORE was developed in two projects funded by the German ministry for education and research (BMBF). In various European-funded projects UNICORE has evolved to a full-grown and well-tested Grid middleware system over the years. UNICORE is used in daily production at several supercomputer centers worldwide. Beyond this production usage, UNICORE serves as a solid basis in many European and international research projects. The UNICORE technology is open source under BSD licence and available at SourceForge, where new releases are published on a regular basis.

The architecture of UNICORE consists of three layers, namely user, server, and target system tier. The user tier is represented by the UNICORE Client, a Graphical User Interface (GUI) that exploits all services offered by the underlying server tiers. Abstract Job Objects (AJO), the implementation of UNICORE's job model concept, are used to communicate with the server tier. An AJO contains platform and site independent descriptions of computational and data related tasks, resource information, and workflow specifications. The sending and receiving of AJOs and attached files within UNICORE is managed by the UNICORE Protocol Layer (UPL) that is placed on top of the Secure Socket Layer (SSL) protocol. The user of an UNICORE Grid does not need to know how these protocols are implemented, as the UNICORE Client assists the user in creating complex, interdependent jobs. For more experienced users a Command Line Interface (CLI) is also available. Both, the UNICORE Client and CLI, provide the functionalities to create and monitor jobs that can be executed on any UNICORE site (Usite) without requiring any modifications, including data management functions like import, export, or transfer of files from one target system to another. In addition, the UNICORE plugin technology allows the creation of application-specific interfaces inside the UNICORE client.

As the single secure entry point the Gateway controls the access to a Usite by accepting and authenticating all requests for server functionality. The Gateway forwards incoming requests to the underlying Network Job Supervisor (NJS) of a Vsite (UNICORE Virtual site) for further processing. A Vsite identifies a particular set of Grid resources at a Usite and is controlled by a NJS. UNICORE supports different system architectures and gives an organization full control over its resources. Vsites may consist of a single supercomputer or a Linux cluster. If more than one resource is operated by an organization there can be one Vsite for each resource inside one Usite. The NJS incarnates an abstract job description received in an AJO to a target system specific job. The target system tier consists of the Target System Interface (TSI), which interfaces with the underlying local resource management system.

The security within UNICORE relies on the usage of permanent X.509 certificates issued by a trusted Certification Authority (CA). These certificates are used to provide a single sign-on in the UNICORE client, i.e. no further password requests are handed to the user. In addition the certificates are used for authentication and authorization, including the mapping of UNICORE user certificates to local accounts, e.g. Unix uid/gid, and for signing AJOs, which are send over SSL based communication channels across 'insecure' internet links. Using X.509 certificates is one example for the consideration of well-known standards, e.g. released by the Global Grid Forum (GGF), within the UNICORE architecture. In addition, UNICORE meets the Open Grid Services Architecture (OGSA) concept following the paradigm of 'everything being a service'.

4. Conclusions.

Interoperability issue in the ACGT project is strongly influenced by architectural model chosen for building ACGT infrastructure. On lower lever layers it is possible to incorporate new components in a transparent way, but changes on higher abstraction levels are much more difficult and would require modification in client interfaces.

Interoperability is also supported by using standards in services implementation. Most of the ACGT services are based on commonly used standards and it make them compliant with other infrastructures.

References

[1] Globus Tooklit http://www.globus.org

[2] Gridge Toolkit http://www.gridge.org

[3] ACGT D4.1 Prototype and report of the ACGT GRID layer

[4] http://www.globus.org/toolkit/docs/4.0/

[5] http://www.globus.org/toolkit/docs/4.0/security/key-index.html

[6] I. Foster, “Globus Toolkit Version 4: Software for Service-Oriented Systems”, IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006

Appendix A - Abbreviations and acronyms

SOA	Service Oriented Architecture
GRMS	Gridge Resource Management System
GAS	Gridge Authorization Service
GDMS	Gridge Data Management System
RFT	Reliable File Transfer
MDS	Monitoring & Discovery Service
WSDL	Web Service Definition Language
SMOA	Smoa Computing - DRMAA access to resources
GRIA	Service-oriented grid infrastructure
BES	Basic Execution Services
GSI	Grid Security Infrastructure
JDSL	Job Submission Description Language
OGF	Open Grid Forum
DRMAA	Distributed Resource Management Application API
BPEL	Business Process Execution Language
WSDL	Web Service Definition Language
OGSA	Open Grid Services Architecture
UNICORE	UNiform Interface to COmputing REsources