Prototype and report of the ACGT GRID layer

Project Number: FP6-2005-IST-026996

Deliverable id: D 4.1

Deliverable name: Prototype and report of the ACGT GRID layer

Submission Date: 03/12/2007

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:	D 4.1
Document name:	Prototype and report of the ACGT GRID layer
Document type (PU, INT, RE)	INT
Version:	2
Submission date:	03/12/2007
Editor: Organisation: Email:	Juliusz Pukacki PSNC pukacki@man.poznan.pl

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

ABSTRACT:

This deliverable presents the Grid infrastructure introduced for ACGT project. Grid Services are placed on two layers on the general software architecture. The first one is responsible for unified access to underlying hardware net of resources distributed all over Europe. The second one consists of more advanced, collective Grid services providing all necessary mechanisms for resource management, data management and grid monitoring.

KEYWORD LIST:Grid, Grid Services, Globus Toolkit, Service Oriented Architecture, Semantic Grid, Grid Resource Management Systems, Data Management, Grid Monitoring

MODIFICATION CONTROL			
Version	Date	Status	Author
1.0	20/11/2007	Draft	J. Pukacki
2.0	03/12/2007	Draft	J.Pukacki
3.0	21/12/2007	Draft	J.Pukacki
4.0	11/01/2008	Draft	J.Pukacki

List of Contributors

- Jarek Nabrzyski, PSNC
- Tomasz Piontek, PSNC
- Bogdan Ludwiczak, 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

1.1. The Grid

At the beginning grid computing was about providing a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. Then it evolved into more general definition of bringing all, different resources together in order to achieve advanced functionality that was not possible without grid. It was defined by Ian Foster et al in "The Anatomy of the Grid: Enabling Scalable Virtual Organizations" as follows:

"The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource brokering strategies emerging in industry, science, and engineering."

The most important attributes of the Grid are:

- coordinates resources that are not subject to centralized control
- using standard, open, general-purpose protocols and interfaces
- to deliver non-trivial qualities of service

The common components existing in Grids are:

- Remote storage and/or replication of data sets
- Publication of datasets using global logical name and attributes in the catalogue
- Security: access authorization and uniform authentication
- Uniform access to remote resources (data and computational resources)
- Publication of services and access cost
- Composition of distributed applications using diverse software components including legacy programs.
- Discovery of suitable datasets by their global logical names or attributes
- Discovery of suitable computational resources
- Mapping and Scheduling of jobs (Aggregation of distributed services)
- Submission, monitoring, steering of jobs execution
- Movement of code/data between the user desktop machines and distributed resources
- Enforcement of quality of service requirements
- Metering and Accounting of resource usage

1.2. The Semantic Web

As described in Wikipedia: *"The Semantic Web is an evolution of the World Wide Web in which information is machine processable (rather than being only human oriented), thus permitting browsers or other software agents to find, share and combine information more* *easily."* In Semantic Web by using additional technology helps to categorize and organize the human information contained on a web page with machine-readable and understandable information that can then be used by applications to process the information. The key technology used are:

- XML
- RDF (Resource Description Framework)
- OWL (Web Ontology Language)

1.3. The Semantic Grid.

The Semantic Grid come into existence as a try to introduce the semantic web technologies into the grids. The semantic grid group describes the semantic grid as *"an extension of the current grid in which information and services are given well-defined meaning, better enabling computers and people to work in cooperation."*

The following graph describes evolution of technologies:

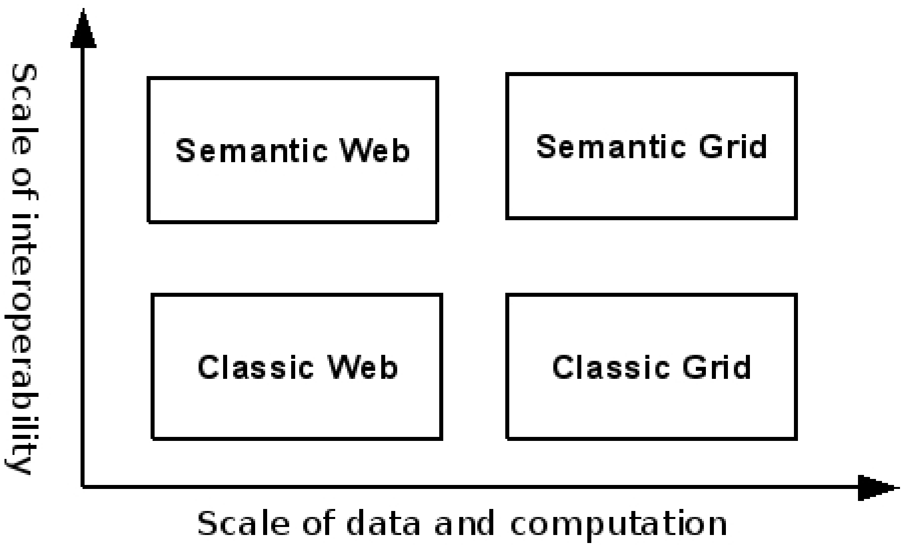


Fig.1. Evolution of technologies

2. Grid Layers on ACGT architecture.

The following picture presents general overview of ACGT architecture.

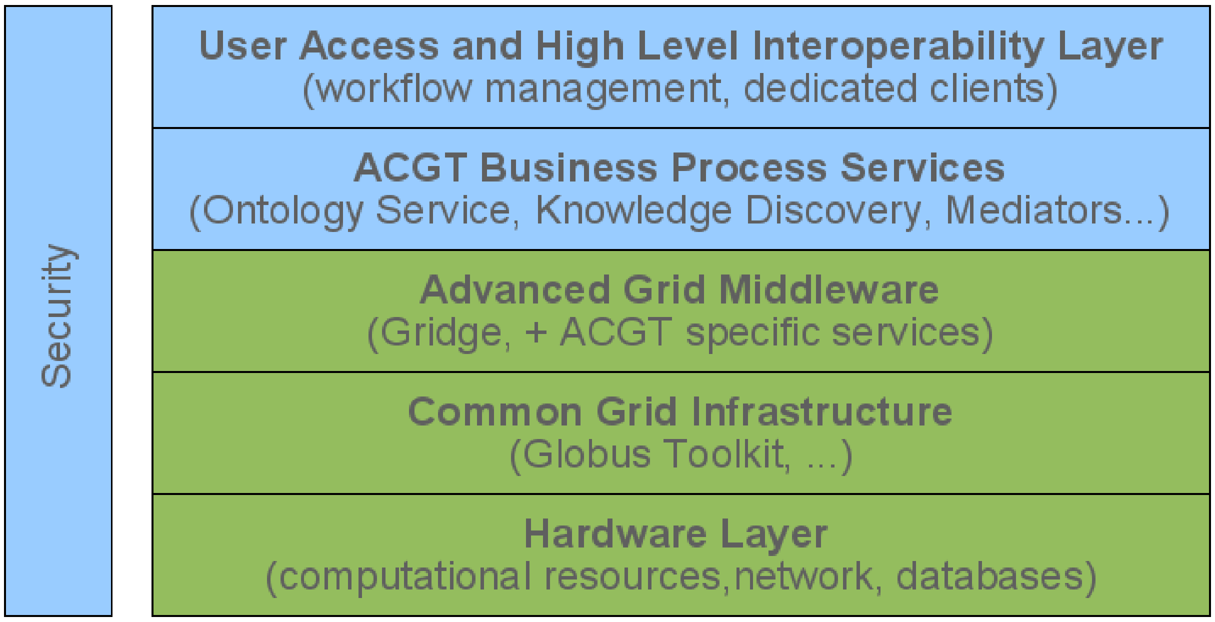

Security
User Access and High Level Interoperability Layer
(workflow management, dedicated clients)
ACGT Business Process Services
(Ontology Service, Knowledge Discovery, Mediators...)
Advanced Grid Middleware
(Gridge, + ACGT specific services)
Common Grid Infrastructure
(Globus Toolkit, ...)
Hardware Layer
(computational resources,network, databases)

Fig.2. ACTT Architecture Overview

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.

3. Basic Grid Services.

Most of the components of this layer come from Globus Toolkit. The newest version of Globus Toolkit (GT4) is used for the ACGT project. as a basic Grid middleware infrastructure.

Globus is not a monolithic software system but consists of some network services that operate together providing unified background for Grid environments.

3.1. GRAM

Grid Resource Allocation Management (GRAM) is a component of the Globus Toolkit responsible for job execution. Grid computing resources are typically operated under the control of a scheduler which implements allocation and prioritization policies while optimizing the execution of all submitted jobs for efficiency and performance. GRAM is not a resource scheduler, but rather a protocol engine for communicating with a range of different local resource schedulers using a standard message format. Rather than consisting of a monolithic solution, GRAM is based on a component architecture at both the protocol and software implementation levels. This component approach serves as an ideal which shapes the implementation as well as the abstract design and features.

There are two implementation of GRAM service: "web services and "pre-web services" Unix server suite to submit, monitor, and cancel jobs on Grid computing resources. Both systems are known under the moniker "GRAM", while "WS GRAM" refers only to the web service implementation.

Job management with GRAM makes use of multiple types of service:

- Job management services represent, monitor, and control the overall job life cycle. These services are the job-management specific software provided by the GRAM solution.
- File transfer services support staging of files into and out of compute resources. GRAM makes use of these existing services rather than providing redundant solutions; WS GRAM has further refactored some file transfer mechanisms present in pre-web service GRAM.
- Credential management services are used to control the delegation of rights among distributed elements of the GRAM architecture based on users' application requirements. Again, GRAM makes use of more general infrastructure rather than providing a redundant solution, and WS GRAM has continued this refactoring to better separate credential management at the protocol level.

For WS GRAM, the Globus Toolkit software development environment, and particularly WSRF core, is used to implement distributed communications and service state. For pre-web service GRAM, the "gatekeeper" daemon and GSI library are used for communications and service dispatch.

WS GRAM utilizes WSRF functionality to provide for authentication of job management requests as well as to protect job requests from malicious interference, while pre-web service GRAM uses GSI and secure sockets directly. The use of GRAM does not reduce the ability for system administrators to control access to their computing resources. The use of GRAM also does not reduce the safety of jobs users run on a given computing resource. To protect users from each other, jobs are executed in appropriate local security contexts, e.g. under specific Unix user IDs based on details of the job request and authorization policies. Additionally, GRAM mechanisms used to interact with the local resource are design to minimize the privileges required and to minimize the risks of service malfunction or compromise. A client may delegate some of its rights to GRAM services in order to facilitate the above functions, e.g. rights for GRAM to access data on a remote storage element as part of the job execution. Additionally, the client may delegate rights for use by the job process itself. With pre-web service GRAM, these two uses of rights are inseparable, while WS GRAM provides separate control for each purpose (while still allowing rights to be shared if that is desired).

WS GRAM provides an "at most once" job submission semantics. A client is able to check for and possibly resubmit jobs, in order to account for transient communication errors without risk of running more than one copy of the job. Similarly, pre-web service GRAM provides a two-phase submission mechanism to submit and then commit a job to be run. While many jobs are allowed to run to their natural completion, GRAM provides a mechanism for clients to cancel (abort) their jobs at any point in the job life cycle.

WS GRAM provides for reliable, high-performance transfers of files between the compute resource and external (GridFTP) data storage elements before and after the job execution. Pre-web service GRAM can also stage with GridFtp systems but with less flexible reliable-transfer logic driving its requests. GRAM supports a mechanism for incrementally transferring output file contents from the computation resource while the job is running. WS GRAM uses a new mechanism to allow arbitrary numbers of files to be transferred in this fashion, while pre-web service GRAM only supports incremental transfer of the job's standard output and error streams.

3.2. MDS

The Monitoring and Discovery System (MDS) is a suite of web services to monitor and discover resources and services on Grids. This system allows users to discover what resources are considered part of a Virtual Organization (VO) and to monitor those resources. MDS services provide query and subscription interfaces to arbitrarily detailed resource data and a trigger interface that can be configured to take action when pre-configured trouble conditions are met. The services included in the WS MDS implementation (MDS4), provided with the Globus Toolkit 4, acquire their information through an extensible interface which can be used to: query WSRF services for resource property information, execute a program to acquire data, or interface with third-party monitoring systems.

Grid computing resources and services can advertise a large amount of data for many different use cases. MDS4 was specifically designed to address the needs of a Grid monitoring system - one that publishes data that is available to multiple people at multiple sites. As such, it is not an event handling system, like NetLogger, or a cluster monitor on its own, but can interface to more detailed monitoring systems and archives, and can publish summary data using standard interfaces.

MDS4 includes two WSRF-based services: an Index Service, which collects data from various sources and provides a query/subscription interface to that data, and a Trigger Service, which collects data from various sources and can be configured to take action based on that data. An Archive Service, which will provide access to historic data, is planned for a future release. The Index Service is a registry similar to UDDI, but much more flexible. Indexes collect information and publish that information as resource properties. Clients use the standard WSRF resource property query and subscription/notification interfaces to retrieve information from an Index. Indexes can register to each other in a hierarchical fashion in order to aggregate data at several levels. Indexes are "self-cleaning"; each Index entry has a lifetime and will be removed from the Index if it is not refreshed before it expires. Each Globus container that has MDS4 installed will automatically have a default Index Service instance. By default, any GRAM, RFT, or CAS service running in that container will register itself to the container's default Index Service. The Trigger Service collects information and compares that data against a set of conditions defined in a configuration file. When a condition is met, or triggered, an action takes place, such as emailing a system administrator when the disk space on a server reaches a threshold.

In addition to the services described above, MDS4 includes several additional software components, including an Aggregator Framework, which provides a unified mechanism used by the Index and Trigger services to collect data. The Aggregator Framework is a software framework used to build services that collect and aggregate data. Services (such as the Index and Trigger services) built on the Aggregator Framework are sometimes called aggregator services, and have the following in common:

- They collect information via Aggregator Sources. An Aggregator Source is a Java class that implements an interface (defined as part of the Aggregator Framework) to collect XML-formatted data
- They use a common configuration mechanism to maintain information about which Aggregator Source to use and its associated parameters (which generally specify what data to get, and from where). The Aggregator Framework WSDL defines an [aggregating service group entry type] that holds both configuration information and data. Administrative client programs use standard [WSRF Service Group registration mechanisms] to register these service group entries to the aggregator service.
- They are self-cleaning - each registration has a lifetime; if a registration expires without being refreshed, it and its associated data are removed from the server.

MDS4 includes the following three Aggregator Sources:

- the Query Aggregator Source, which polls a WSRF service for resource property information,
- the Subscription Aggregator Source, which collect data from a WSRF service via WSRF subscription/notification,
- the Execution Aggregator Source, which executes an administrator-supplied program to collect information.

Depending on the implementation, an Aggregator Source may use an external software component (for example, the Execution Aggregator Source uses an executable program), or a WSRF service may use an external component to create and update its resource properties (which may then be registered to an Index or other aggregator service, using the Query or Subscription Aggregator Source). We refer to this set of components as Information Providers. Currently, MDS4 includes the following sources of information: Hawkeye Information Provider, Ganglia Information Provider, WS GRAM, Reliable File Transfer Service (RFT), Community Authorization Service (CAS) , any other WSRF service that publishes resource properties

3.3. RFT

RFT is a Web Services Resource Framework (WSRF) compliant web service that provides "job scheduler"-like functionality for data movement. You simply provide a list of source and destination URLs (including directories or file globs) and then the service writes your job description into a database and then moves the files on your behalf. Once the service has taken your job request, interactions with it are similar to any job scheduler. Service methods are provided for querying the transfer status, or you may use standard WSRF tools (also provided in the Globus Toolkit) to subscribe for notifications of state change events. We provide the service implementation which is installed in a web services container (like all web services) and a very simple client. There are Java classes available for custom development, but due to lack of time and resources, work is still needed to make this easier.

3.4. GridFTP

GridFTP is a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. The GridFTP protocol is based on FTP, the highly-popular

Internet file transfer protocol. We have selected a set of protocol features and extensions defined already in IETF RFCs and added a few additional features to meet requirements from current data grid projects.

GridFTP functionality includes some features that are supported by FTP extensions that have already been standardized (RFC 959) but are seldom implemented in current systems. Other features are new extensions to FTP.

- Grid Security Infrastructure and Kerberos support: Robust and flexible authentication, integrity, and confidentiality features are critical when transferring or accessing files. GridFTP must support GSI and Kerberos authentication, with user controlled setting of various levels of data integrity and/or confidentiality. GridFTP implements the authentication mechanisms defined by RFC 2228, "FTP Security Extensions".
- Third-party control of data transfer: To manage large datasets for distributed communities, we must provide authenticated third-party control of data transfers between storage servers. A third-party operation allows a user or application at one site to initiate, monitor and control a data transfer operation between two other sites: the source and destination for the data transfer. Our implementation adds Generic Security Services (GSS)-API authentication to the existing third-party transfer capability defined in the FTP standard.
- Parallel data transfer: On wide-area links, using multiple TCP streams in parallel (even between the same source and destination) can improve aggregate bandwidth over using a single TCP stream. GridFTP supports parallel data transfer through FTP command extensions and data channel extensions.
- Striped data transfer: Data may be striped or interleaved across multiple servers, as in a DPSS network disk cache. GridFTP includes extensions that initiate striped transfers, which use multiple TCP streams to transfer data that is partitioned among multiple servers. Striped transfers provide further bandwidth improvements over those achieved with parallel transfers. We have defined GridFTP protocol extensions that support striped data transfers.
- Partial file transfer: Some applications can benefit from transferring portions of files rather than complete files: for example, high-energy physics analyses that require access to relatively small subsets of massive, object-oriented physics database files. The best that the standard FTP protocol allows is transfer of the remainder of a file starting at a particular offset. GridFTP provides commands to support transfers of arbitrary subsets or regions of a file.
- Automatic negotiation of TCP buffer/window sizes: Using optimal settings for TCP buffer/window sizes can dramatically improve data transfer performance. However, manually setting TCP buffer/window sizes is an error-prone process (particularly for non-experts) and is often simply not done. GridFTP extends the standard FTP command set and data channel protocol to support both manual setting and automatic negotiation of TCP buffer sizes for large files and for large sets of small files.
- Support for reliable and restartable data transfer: Reliable transfer is important for many applications that manage data. Fault recovery methods are needed to handle failures such as transient network and server outages. The FTP standard includes basic features for restarting failed transfers that are not widely implemented. GridFTP exploits these features and extends them to cover the new data channel protocol.

4. Advanced Grid Middleware.

The components of this layer are serving more advanced mechanisms in The Grid. They can be called collective services because they operate on many lower level resources. In most cases they are not required for the environment but they increase quality of service and can serve more advanced scenarios for the Grids.

4.1. GRMS

The Gridge Resource Management System (GRMS) is an open source meta-scheduling system, which allows developers to build and deploy resource management systems for large scale distributed computing infrastructures. The GRMS, based on dynamic resource selection, mapping and advanced scheduling methodology, combined with feedback control architecture, deals with dynamic Grid environment and resource management challenges, e.g. load-balancing among clusters, remote job control or file staging support. Therefore, the main goal of the GRMS is to manage the whole process of remote job submission to various batch queuing systems, clusters or resources. It has been designed as an independent core component for resource management processes which can take advantage of various low-level Core Services and existing technologies. Finally, the GRMS can be considered as a robust system which provides abstraction of the complex grid infrastructure as well as a toolbox which helps to form and adapts to distributing computing environments.

The GRMS is a central point for all resource and job management activities an tightly cooperates with other services: authorization, information systems, monitoring, data management to fulfil applications requirements. The main features of the GRMS are:

- job submission
- job control (suspending, resuming, cancelling)
- ability to chose "the best" resource for the job execution using multicriteria matching algorithm
- support for the job checkpointing and migration
- support for file staging
- storing all information about the job execution
- user notifications support
- workflow jobs support

The GRMS has been designed as an independent set of components for resource management processes which can take advantage of various low-level Core Services, e.g. GRAM, GridFTP and Gridge Monitoring System, as well as various grid middleware services, e.g. Gridge Authorization Service, Gridge Data Management Service. All these services working together provide a consistent, adaptive and robust grid middleware layer which fits dynamically to many different distributing computing infrastructures. The GRMS implementation requires Globus software to be installed on grid resources, and uses Globus Core Services deployed on resources: GRAM, GridFtp, MDS (optional). The GRMS supports Grid Security Infrastructure by providing the GSI-enabled web service interface for all clients, e.g. portals or applications, and thus can be integrated with any other middleware grid environment.

One of the main assumptions for the GRMS is to perform remote jobs control and management in the way that satisfies Users (Job Owners) and their applications requirements. All users requirements are expressed within XML-based resource specification documents and sent to the GRMS as SOAP requests over GSI transport layer connections.

Simultaneously, Resource Administrators (Resource Owners) have full control over resources on which all jobs and operations will be performed by appropriate GRMS setup and installation. Note, that the GRMS together with Core Services reduces operational and integration costs for Administrators by enabling grid deployment across previously incompatible cluster and resources. Technically speaking, the GRMS is a persistent service within a Tomcat/Axis container. It is written completely in Java so it can be deployed on various platforms.

GRMS supports Grid Security Infrastructure by providing GSI-enabled Web Service interfaces and in fact acts on behalf of end users. The communication between the GRMS service and all clients is done through a GSI-enabled HTTP-based protocol called httpg implementing transport-level security introduced by Globus community. With the GAS, GRMS is able to manage both, job grouping and jobs within collaborative environments according to predefined VO security rules and policies. With the Data Management services from Gridge, GRMS can create and move logical files/catalogs and deal with data intensive experiments. Gridge Monitoring Service can be used by GRMS as an additional information system. Finally, Mobile service can be used to send notifications via SMS/emails about events related to users jobs and as a gateway for GRMS mobile clients.

GRMS is able to store all operations in a database. Based on this information a set of very useful statistics for both end users and administrators can be produced. All the data are also a source for further, more advanced analysis and reporting tools. All users preferences and job requirements must be expressed as XML-based resource specification documents, called GRMS Job Description. Once such a request is sent, each job within GRMS receives a unique ID and the whole process of job scheduling and control begins.

GRMS has modular architecture, it consists of some functional modules (Fig.3)

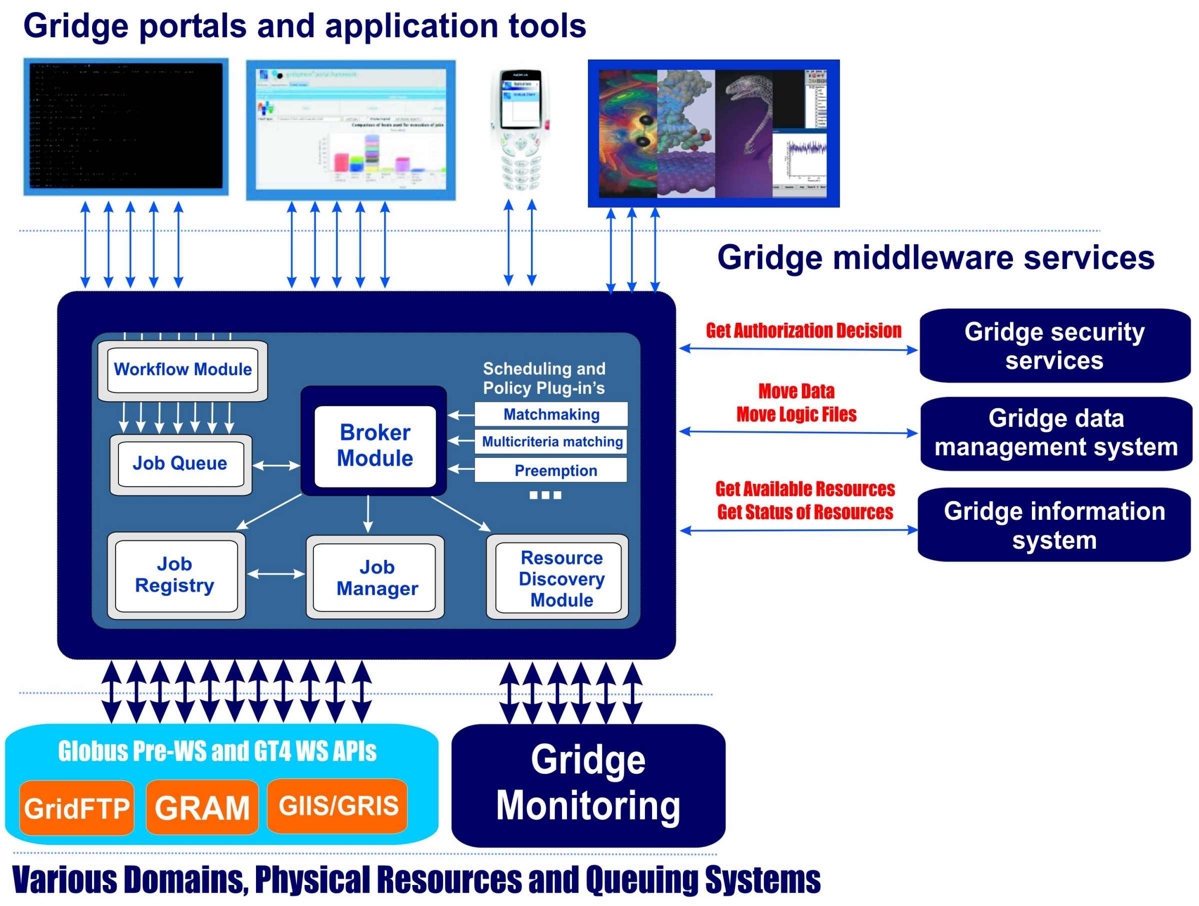

Gridge portals and application tools
Gridge middleware services
Workflow Module
Scheduling and Policy Plug-in's
Matchmaking
Multicriteria matching
Preemption
Broker Module
Job Queue
Job Registry
Job Manager
Resource Discovery Module
Get Authorization Decision
Gridge security services
Move Data
Move Logic Files
Gridge data management system
Get Available Resources
Get Status of Resources
Gridge information system
Globus Pre-WS and GT4 WS APIs
GridFTP
GRAM
GIIS/GRIS
Gridge Monitoring
Various Domains, Physical Resources and Queuing Systems

Fig.3. GRMS Architecture

1) Broker Module

- Steering process of job submission,
- Choosing the best resources for job execution (scheduling algorithm),
- Transferring input and output files for job's executable.

2) Resource Discovery Module

- Finding resources that fulfils requirements described in Job Description,
- Providing information about resources, required for job scheduling.

3) Job Manager Module

- Checking status of running job,
- Cancelling running job,
- Monitoring for status changes of running job.

4) Job Queue

- Providing internal job queuing,
- Providing ability to implement different queue management algorithms.

5) Web Service Interface

- Providing user access by HTTPS or HTTPG protocols.
- Communicating with Broker Module by RMI protocol.
- Returning job information without Broker Module mediation.

6) Job Registry

- Providing persistent storage of internal job data.
- Contains job execution history.

7) Workflow Module

- Providing workflow tasks execution.
- Triggering job's tasks due to workflow description.

The aim of the Broker Module is to control the whole process of resource and job management within GRMS. This module steers a flow of requests to GRMS and is also responsible for an appropriate cooperation with other modules. In the current release of GRMS, the Broker Module contains basic scheduling and policy strategies: matchmaking and multi-criteria matchmaking. The first strategy is relatively simple but in fact very efficient approach for managing resources on which the advanced reservation is not possible. The second strategy allows more flexible and more accurate resource selections according to both users and administrator's requirements and preferences. These two strategies can be easily modified as well as new scheduling and policy modules can be integrated with the Broker Module.

The Job Manager Module is responsible for monitoring of job status changes within GRMS and then storing information in a Job Registry together with many additional parameters (including resource requirements of jobs, user names, job Ids, submission times, pending times, execution times, jobmanagers to which jobs were submitted, history of migration if jobs have been migrated, etc). Due to an importance of historic information, especially in multi site or large scale resource management systems, GRMS provides the interface for users and administrator to receive information about past GRMS activities. The access to historic information can be considered as a feedback to the Broker Module and allows developers to implement new dynamic scheduling approaches and policies within GRMS, e.g. prediction based schedulers or fair-sharing. The tracking of historical resource utilization for all users results in the ability to modify job priorities, ensuring a balance appropriate access, and optimizing administrator criteria (e.g. job throughput or turnaround time).

The Resource Discovery Module monitors a status of distributed resources and therefore uses a flexible hierarchical access to both central and local information services. This module uses various techniques to discover and get an efficient access to up-to-date and accurate (both static and dynamic) information about resources. The goal of the Resource Discovery Module it to deliver all information in a form and in time required by the Broker Module and its scheduling and policy strategies. Also to this module new extending techniques can be applied (e.g. indexing or caching) to speed up a flow of information.

Job Registry stores information about jobs and tasks in RDBMS. These data contains informations about job's current status, description, coallocation etc. Due to speed improvement, job's database is divided on two parts: in first of them only data about active (unfinished) jobs is stored, in second one data about already finished jobs.

Workflow Engine is a complete, robust and full featured workflow management engine, responsible for steering of the execution of the particular tasks of the workflow. It keeps track of the running tasks, triggers the execution of the next ones according to defined workflow. It also keeps consistency of the input and output files that might be just references inside the workflow.
Task triggering in a workflow is based on the status changes of the parent task(s) - any state can be defined as a trigger. So it is possible to define a child task that will be executed as soon as the parent for instance fails. Besides it the case of more than one parents it is possible to specify if the triggering event is alternative or conjunction of the parent's statuses changes or. One of the other important role of the Workflow Engine is the support for the parametric job descriptions - it decomposes the parametric description into bunch of separate tasks and manages its execution.

Job Queue is a fully configurable, statefull repository of the submitted to GRMS jobs and tasks. It is possible to define different algorithm for selecting tasks for the execution. Default configuration is implemented as a FIFO queue. The interesting mechanism is ability to configure the number of tasks served simultaneously which is very useful in case of deploying GRMS on slower machines.

4.2. GAS

The Gridge Authorization Service (GAS) is an authorization system which can be the standard decision point for all components of a system. Security policies for all system components are stored in GAS. Using these policies GAS can return an authorization decision upon the client request. GAS has been designed in such a way that it is easy to perform integration with external components and it is easy to manage security policies for complex systems. The possibility to integrate with the Globus Toolkit and many operating system components makes GAS an attractive solution for grid applications.

Generally, an authorization service can be used for returning an authorization decision upon the user request. The request has to be described by three attributes: user, object and operation. The requester simply asks if the specific user can perform the operation on the specific object. Obviously, the query to an authorization service can be more complex and the answer given by such service can be complicated, too. One of the services which can work in such scenario is the Gridge Authorization Service (GAS). GAS has been designed in a form which enables many possible applications. GAS can communicate in many ways with other components. By using the modular structure of GAS it is easy to write a completely new communication module. The GAS complex data structure can be used to model many abstract and real world objects and security policies for such objects. For example, GAS has been used for managing security policies: for many Virtual Organizations, for services (like Gridge Resource Management Service, iGrid, Mobile Services and other) and for abstract objects like communicator conferences or computational centers. These and many other features give a possibility to integrate GAS with many existing solutions. Such integration can be very important, because it raises the security level of the existing solutions and makes it possible to use the newest security technologies.

The main goal of GAS is to provide a functionality that would be able to fulfill most authorization requirements of grid computing environments. GAS is designed as a trusted single logical point for defining security policy for complex grid infrastructures. As flexibility is the key requirement, it is to be able to implement various security scenarios, based on push or pull models, simultaneously. Secondly, GAS is considered as independent of specific technologies used at lower layers, and it should be fully usable in environments based on grid toolkits as well as other toolkits. The high level of flexibility is achieved mainly through the modular design of GAS and usage of a complex data structure which can model many scenarios and objects from the real world. It means that GAS can use many different ways for communication with external components and systems, use many security data models and hold security policy on different types of storage systems. These features make GAS attractive for many applications and solutions (not only for those related with grids). GAS has to be the trusted component of each system in which it is used and it brings about that the implementation of GAS was written in ANSI C. This choice makes GAS a very fast and stable component which uses not much CPU power and little amount of memory. The main problem of many authorization systems is their management. It is not easy to work with a complex system in a user-friendly way. Based on many experiences and the end user comments together with GAS, the GAS administration portlet (web application) is provided, which makes management as easy as possible. Flexibility of this solution gives users a full possibility of presenting only these security policies which are important for them. The GAS management is possible in two other ways: by the GUI GTK client and by the command line client.

4.3. GDMS

Data storage, management and access in Gridge environment is supported by the Gridge Data Management Suite (DMS). This suite composed of several specialized components allows to build a distributed system of services capable of delivering mechanisms for seamless management of large amount of data. This distributed system is based on the pattern of autonomic agents using the accessible network infrastructure for mutual communication. From the external applications point of view DMS is a virtual file system keeping the data organized in a tree structure. The main units of this structure are metadirectories, which enable to put a hierarchy over other objects and metafiles. Metafiles represent a logical view of computational data regardless of their physical storage location.

Data Management System consists of three logical layers: the Data Broker, which serves as the access interface to the DMS system and implement the brokering of storage resources, the Metadata Repository that keeps information about the data managed by the system, and the Data Container, which is responsible for the physical storage of data. In addition, DMS contains modules which extend its functionality to fulfill the enterprise requirements. These include the fully functional web based administrator interface and a Proxy to external scientific databases. The Proxy provides a SOAP interface to the external databases, such as for example those provided by SRS (Sequence Retrieval System).

The Data Broker is designed as an access point to the data resources and data management services. A simple API of the Data Broker allows to easily access the functionality of the services and the stored data. The Data Broker acts as a mediator in the flow of all requests coming from external services, analyzes them and eventually passes to the relevant module. The DMS architecture assumes that multiple instances of the Data Broker can be deployed in the same environment, thus increasing the efficiency of data access from various points in the global Grid environment structure.

The Metadata Repository is the central element of the Gridge distributed data management solution. It is responsible for all metadata operations as well as their storage and maintenance. It manages metadata connected with the data files, their physical locations and transfer protocols that could be used to obtain them, with the access rights to the stored data and with the metadescriptions of the file contents. Currently each DMS installation must contain a single instance of the Metadata Repository, which acts as a central repository of the critical information about the metacatalogue structure, user data and security policy for the whole DMS installation.

The Data Container is a service specialized towards the management of physical data locations on the storage resources. The Data Container API is designed in a way to allow easy construction and participation in the distributed data management environment of storage containers for different storage environments. The Data Containers currently available in the DMS suite include a generic file system Data Container, a relational database Data Container and a tape archiver Data Container. The data stored on the various storage resources can be accessed with one of the many available protocols including such as GASS, FTP and GridFTP.

The Proxy modules are services that join the functionality of the Metadata Repository allowing to list the available databanks, list their content, read the attached metadata attributes and to build and execute queries, and of the Data Container to provide the data using the selected data transfer protocol. Such Proxy container are highly customized towards the specific platform they are working with to allow building complex queries and executing operations on the found entries.

4.4. Mercury - Grid Monitoring

The Mercury Grid Monitoring System has been developed within the GridLab project. It provides a general and extensible Grid monitoring infrastructure. Mercury Monitor is designed to satisfy specific requirements of Grid performance monitoring: it provides monitoring data represented as metrics via both pull and push model data access semantics and also supports steering by controls. It supports monitoring of Grid entities such as resources and applications in a generic, extensible and scalable way.

The Mercury Monitoring is designed to satisfy requirements of Grid performance monitoring: it provides monitoring data represented as metrics via both pull and push access semantics and also supports steering by controls. It supports monitoring of Grid entities such as resources and applications in a generic, extensible and scalable way. It is implemented in a modular way with emphasis on simplicity, efficiency, portability and low intrusiveness on the monitored system.

The aim of the Mercury Monitoring system is to support the advanced scenarios in Grid environment, such as application steering, self-tuning applications and performance analysis and prediction. To achieve this the general GGF GMA architecture is extended with actuators and controls. Actuators are analogous to sensors in the GGF GMA but instead of gathering information, they implement controls and provide a way to influence the system.

The architecture of Mercury Monitor is based on the Grid Monitoring Architecture (GMA) proposed by Global Grid Forum (GGF), and implemented in a modular way with emphasis on simplicity, efficiency, portability and low intrusiveness on the monitored system.

The input of the monitoring system consists of measurements generated by sensors. Sensors are controlled by producers that can transfer measurements to consumers when requested.

Sensors are controlled by producers that can transfer measurements to consumers when requested. Sensors are implemented as shared objects that are dynamically loaded into the producer at run-time depending on configuration and incoming requests for different measurements.

In Mercury all measurable quantities are represented as metrics. Metrics are defined by a unique name such as host.cpu.user which identifies the metric definition, a list of formal parameters and a data type. By providing actual values for the formal parameters a metric instance can be created representing an entity to be monitored. A measurement corresponding to a metric instance is called metric value.

Metric values contain a time-stamp and the measured data according to the data type of the metric definition. Sensor modules implement the measurement of one or more metrics. Mercury Monitor supports both event-like (i.e. an external event is needed to produce a metric value) and continuous metrics (i.e. a measurement is possible whenever a consumer requests it such as, the CPU temperature in a host).

Continuous metrics can be made event-like by requesting automatic periodic measurements. In addition to the functionality proposed in the GMA document, Mercury also supports actuators.

Actuators are analogous to sensors but instead of taking measurements of metrics they implement controls that represent interactions with either the monitored entities or the monitoring system itself. In addition to all mentioned features Mercury facilitates steering.

4.5. Mobile User Services

Mobile software development in Gridge is focused on providing a set of applications that would enable communication between Mobile devices, such as cell phones, Personal Digital Assistants (PDA) or laptops and Grid Services on the other side. This class of applications is represented by clients running on mobile devices, mobile gateways acting as a bridge between clients and Grid services as well as additional specialized middleware services for mobile users.

The main goal of the services is to make use of small and flexible mobile devices that are increasingly used for web access to various remote resources. The system provides Grid access mechanisms for such devices. This requires adoption of the existing access technologies like portals for low bandwidth connectivity and low level end-user hardware. The mobile nature of such devices also requires flexible session management and data synchronization. The system enhances the scope of present Grid environments to the emerging mobile domain. Utilizing new higher bandwidth mobile interconnects, very useful and previously impossible scenarios of distributed and collaborative computing can be realized. To achieve this and taking into consideration some still existing constraints of mobile devices, the Access for Mobile Users group is developing a set of applications in the client-server model with the J2ME CLDC/MIDP- java client, and portlet server working with GridSphere. This set allow us to manage end user Grid jobs (steer an application) or view messages and visualizations produced by Grid applications on device such simple as standard mobile phone. The second group of developed services is tightly connected with end user notifications about various events in Grids. Events like: the information about user application is started or finished, the visualization is ready for viewing or waiting for new data, can be send to end users using various notifications way. It can be Email, SMS, MMS, or message of one of Internet Communicators like AIM, Yahoo, ICQ, Jabber etc. (including most popular in Poland Gadu-Gadu and Tlen).

Mobile services gives also end users possibility to start a conference concerning aforementioned event between users of given virtual organization (including conferences between clients of different communicators).

The unique possibility of giving access to Grid resources for users of relatively weak devices is one of features that distinguish Gridge mobile applications from other Grid systems. Moreover, the used technology, Java 2 Micro Edition - Mobile Information Device Profile (J2ME-MIDP) applications (midlets) on the client side allows to develop flexible, possibly off-line working programs that may be used on a wide range of devices supporting J2ME. Using the MIDP compliant device internal repository for storing data, gives the user possibility to use it later in offline state and prepare the data, to be sent in on-line state.

The Mobile Command Center (MCC) that acts as a gateway between mobile client and Grid services is developed in Java as a GridSphere portlet (see Gridsphere.org) with separate "mobile" context. MCC automatically grabs the device profile (like device class, screen size, color depth, etc), this information is used during forwarding the request from mobile device to Grid services (mainly GSI-enabled Web Services like Gridge MessageBox, Visualization Service for Mobiles or Gridge Resource Management System). Services that can be accessed from mobile device using MCC belong to two groups: the first group consists of Grid services that were adopted to use with mobile devices, the second group are services developed for use only with mobile devices. The Visualization Service for Mobiles belong to second group and is used to view the application output in form of visualization prepared exactly according to the User's device capabilities. The advantage in this case is as follows: the large amount of data is not sent via weak GPRS connections to the device that cannot store it in the memory and cannot display it correctly. First group of services consists of Gridge Resource Management System and Notification and Messenger Service. The first service can be used in "Collaborative scenario" - the user can steer the application (even not being an owner) from mobile device. He/she can get the jobs list, migrate, resume, suspend, cancel, edit, view history and submit new job on the basis of edited/modified description of already finished jobs. Using GRMS together with Notification service the user can register for user notifications related to the running jobs. In this way the user is notified about important events occurring in the Grid (like jobs status changes, application output availability). These notifications can be send as Email,SMS and Internet Communicator (AIM, Yahoo etc) messages to the user. Using the Messenger Service it is possible also to make a conference between users of Virtual Organization defined in Gridge Authorization Service even if they use different communicators.

4.6. OGSA-DAI

OGSA-DAI is a middleware product which allows data resources, such as relational or XML databases, to be exposed on to Grids. Various interfaces are provided and many popular database management systems are supported. The software also includes a collection of components for querying, transforming and delivering data in different ways, and a simple toolkit for developing client applications. OGSA-DAI is designed to be extensible, so users can add their own additional functionality.

The architecture of OGSA-DAI consists of a five layers each serving a different purpose. The lowest layer is Data Layer. It consists of the data resources that can be exposed via OGSA-DAI. Currently these include:

- Relational databases such as MySQL, SQL Server, DB2, Oracle, PostgreSQL,
- XML databases such as eXist, Xindice
- Files and directories in formats such as OMIM, SWISSPROT and EMBL

Business Logic Layer layer encapsulates the core functionality of OGSA-DAI. It consists of components known as data service resources. Multiple data service resources can be deployed to expose multiple data resources. There is a 1-1 relationship between data service resources and data resources. The responsibilities of a data service resource include:

- Execution of perform documents - a perform document describes the actions that a data service resource should to take on behalf of the client. Each action is known as an activity. OGSA-DAI already includes a large number of activities for performing common operations such as database queries, data transformations and data delivery.
- Generation of response documents - a response document describes the status of execution of a perform document and may contain result data, such as the results from a database query.
- Data resource access - interactions with data resources take place via the data resource accessor component.
- Data transport functionality - data can be streamed in and out of data service resources to and from clients and other data service resources.
- Session management - the creation, access and termination of session objects allowing state to be stored across multiple requests to the data service resource. All perform document requests are processed within a session. Sessions are also used for storing the streams used by the data transport functionality. These are known as session streams.
- Property management - the creation, access and removal of properties associated with the data service resource. These are known as data service resource properties and are generally used for exposing meta-data such as the status of a request or the schema of the underlying data resource.

Presentation Layer encapsulates the functionality required to expose data service resources using web service interfaces. OGSA-DAI includes two realisations, one compliant with WSRF and the other compliant with WSI (that is a solution that only relies on the specifications mentioned in the WS-I basic profile, i.e. that do not use WSRF). For each realisation there is a WSDL document that describes the interface.

A client can interact with a data service resource via a corresponding data service. Depending on whether a WSRF or WSI data service has been deployed, the client application must be compliant n must be compliant with the WSRF or WSI standards.

5. Services Classification.

In the ACGT environment there can be a lot of different services. The SOA paradigm defines architecture as a loosely coupled software services to support the requirements of business processes and software users. That is very general definition. We would like to build ACGT environment in a SOA manner but it is required to add some semantic, some additional constraints and rules for building the ACGT services. The services in ACGT environment can be categorized based on different criteria:

- Position in ACGT architecture

  The goal of layered architecture is to introduce different abstractions levels for services. Services from different layers operates on different terminology. Some of them are located near the physical resources using hardware terms, the others ale located near end user and should be contacted using language of meta descriptions. The other important fact is that upper layer services are specific to some scientific area and lower level ones are more general and could be used in a generic way by different clients:

  - Service of Common Grid Layer

  The Services from that layer are used for accessing hardware resources.

  - Service of Advanced Grid Layer

  Provides more advanced, collective functionality, using lower level services to realize clients requests.

  - Service of Business Model Layer

  These are specific services for ACGT environment. They are closer to the end user, can operate on terms from bio and cancer research world (meta descriptions, ontology)

- Role in ACGT environment

  It is obvious that services can play different roles in distributed environment. For ACGT we can define two main groups:

  - Infrastructure Service

  The services used for building ACGT environment (middleware), they are performing some more general actions - management issues. Most of the services from Grid layers belongs to this group. The examples of such a service are: Registry Service, Resource Management, etc...

  - Analytical Tool Service

  Tools for concrete, bio-oriented tasks. They are mostly located in ACGT Business Model Layer

- Owner/maintainer of the service

  - ACGT Service

  Service developed or adopted by ACGT consortium that has full control over it. The control means possibility to introduce common policies for building such a service, for instance applying the same security mechanisms, the same technology for accessing the service, etc. This services should also be trustworthy - we need to be sure that what the service is supposed to do is in fact doing.

  Remark: all Infrastructure Services should be ACGT Services.

- Third Party Service

  It is not possible to build everything from scratch. There is a lot of existing tools available freely that can be used without any limitations. We should to define the policy (evaluation procedures) of incorporating such services to ACGT infrastructure. In some cases it means developing additional components - wrappers etc.

Consequences of that classification.

- Security

  Classification of the services is very important from a security point of view. It is necessary for defining policies for creation, introducing or managing the services. The point is: there will be different policies for different types of services (this is main motivation of classification).

  For example: all Infrastructure Services should use the same authentication/authorization mechanisms, and after validation procedures can be treated as a trusted services.

  On the other hand it is not possible to enforce any implementation changes to Third Party Services so we need to remember that it is forbidden to send any confidential data to them.

  Analytical Tools developed by ACGT members should also obey the rules for authentication/authorization but we can leave some of them publicly available (without any access restrictions).

- Integration

  From the point of view of integration for each group of services we can define what technology we support. For ACGT services we can enforce common technology of implementation (that would be useful for Infrastructure Services)

6. Grid services implementation.

6.1. Introduction

This section contains all the necessary information to implement and deploy a secure ACGT web service from scratch based on the Globus Toolkit (version 4.1.2).

6.2. Preparing Hosting Environment (Service Container)

The hosting environment for the axis based web service is tomcat as a servlet container and axis as a SOAP engine. The simplest way to create the hosting environment for gsi-enabled web services is to deploy the java ws-core (which is part of Globus Toolkit 4) into tomcat and then to remove or not unnecessary globus core WSRF services leaving only part of globus responsible for ws-security.

6.3. Creating a low privilege account

The first [optional] step is to create a low privilege user for the service. On most linux/unix systems the following commands should work.

```
groupadd services
useradd -g services -d /home/hello -c "GSI-enabled Hello World Web Service"
hello
```

6.4. Setup CA certificates

This step defines set of Cetificate Authorities which the service will trust.

1.Download the ACGT CA certificates: Acgt-ca.tar

2.Extract the certificates and place them in the correct directory

- $HOME/.globus - to be visible only for the created user,
- /etc/grid-security - to be globally visible.

```
cd $HOME/.globus
tar xpf ~/Acgt-ca.tar
chown -R hello:services *
```

6.5. Requesting service certificate

- use the grid-cert-request command to create a certificate request for the service. The grid-cert-request command is part of Globus toolkit.

```
export X509_CERT_DIR=/home/piontek/.globus/certificates/; grid-cert-request
-service hello -host druid-bis.man.poznan.pl -ca -dir .
```

- send the certificate request (*hellocert_request.pem* file) to the ACGT CA and wait for the signed certificate.

6.6. Downloading and installing Globus Toolkit 4.1.2

You can skip this point if the GT 4.1.2 is already installed.

```
wget http://www-unix.globus.org/ftppub/gt4/4.1.2/installers/src/
gt4.1.2-all-source-installer.tar.gz
tar xzf gt4.1.2-all-source-installer.tar.gz
mkdir gt-4.1.2
cd gt4.1.2-all-source-installer
```

Use --prefix option to choose destination directory.

```
./configure --prefix=/home/piontek/tmp/ACGT/globus/gt-4.1.2
```

Globus Toolkit 4.1.2 can be also downloaded from alternative site.

Install Globus limiting installation only to wsjava package (aka. ws-core).

```
make wsjava install
```

If the installation fails with following of similar error:

```
~/tmp/ACGT/globus/gt4.1.2-all-source-installer $ make wsjava install
cd gpt && OBJECT_MODE=32 ./build_gpt
build_gpt ====> installing GPT into /home/piontek/tmp/ACGT/globus/gt-4.1.2
build_gpt ====> building /home/piontek/tmp/ACGT/globus/gt4.1.2-all-source-
installer/gpt/support/Compress-Zlib-1.21
build_gpt ====> building /home/piontek/tmp/ACGT/globus/gt4.1.2-all-source-
installer/gpt/support/IO-Zlib-1.01
build_gpt ====> building /home/piontek/tmp/ACGT/globus/gt4.1.2-all-source-
installer/gpt/support/makepatch-2.00a
build_gpt ====> building /home/piontek/tmp/ACGT/globus/gt4.1.2-all-source-
installer/gpt/support/Archive-Tar-0.22
build_gpt ====> building /home/piontek/tmp/ACGT/globus/gt4.1.2-all-source-
installer/gpt/support/PodParser-1.18
build_gpt ====> building /home/piontek/tmp/ACGT/globus/gt4.1.2-all-source-
installer/gpt/support/Digest-MD5-2.20
build_gpt ====> building /home/piontek/tmp/ACGT/globus/gt4.1.2-all-source-
installer/gpt/packaging_tools
/home/piontek/tmp/ACGT/globus/gt-4.1.2/sbin/gpt-build   -srcdir=source-
trees/wsrf/java/common/source gcc32dbg
sh: NOT: command not found
ERROR: Untar failed
make: *** [globus_java_ws_core_common] Error 255
```

Plese check if locations of all needed tools in the *${GLOBUS_PATH}/var/lib/perl/Grid/GPT/LocalEnv.pm* file are set properly.

6.7. Downloading and installing Apache Tomcat

Download and install Apache Tomcat servlet container (version 5.5.x)

```
wget http://ftp.tpnet.pl/vol/d1/apache/tomcat/tomcat-5/v5.5.25/bin/
apache-tomcat-5.5.25.tar.gz
tar xzf apache-tomcat-5.5.25.tar.gz
```

Please note, that Tomcat 5.5.x requires java JRE 5.0. If you want to use java 1.4 please install additionally the JDK 1.4 Compatability Package

The Tomcat 5.5.25 and JDK 1.4 Compatability Package can be also downloaded from alternative site.

6.8. Deploying wsjava into Tomcat

Go to the directory where the wsjava package was installed and do following set of actions:

```
$ export GLOBUS_LOCATION=`pwd`
$ ant -f share/globus_wsrf_common/tomcat/tomcat.xml deploySecureTomcat
-Dtomcat.dir=<tomcat.dir>
```

Where <tomcat.dir> is an absolute path to the Tomcat installation directory.

Also, -Dwebapp.name=<name> property can be specified to set the name of the web application under which the installation will be deployed. By default "wsrf" web application name is used. In our example the name was set to acgt value.

```
$ export GLOBUS_LOCATION=`pwd`
$ ant -f share/globus_wsrf_common/tomcat/tomcat.xml deploySecureTomcat
-Dtomcat.dir=<tomcat.dir> -Dwebapp.name=acgt
```

6.9. Configuring Tomcat

In addition to the above deployment step it is also needed to configure the Tomcat to use appropriate connectors and valves responsibles for GSI-security. Please modify the <tomcat.dir>/conf/server.xml configuration file.

- Add a HTTPS Connector in the <Service name="Catalina"> section and update the parameters appropriately with your local configuration:

```
<Connector
     className="org.globus.tomcat.coyote.net.HTTPSConnector"
     port="8443" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
     autoFlush="true"
     disableUploadTimeout="true" scheme="https"
     enableLookups="true" acceptCount="10" debug="0"
     protocolHandlerClassName="org.apache.coyote.http11.Http11Protocol"
     socketFactory="org.globus.tomcat.catalina.net.BaseHTTPSServerSocketFac
tory"
     proxy="/path/to/proxy/file"
     cert="/path/to/certificate/file"
     key="/path/to/private/key/file"
     cacertdir="/path/to/ca/certificates/directory"
     encryption="true"/>
```

In the above the proxy, cert, key and cacertdir attributes are optional. Furthermore, the proxy and the combination of cert and key attributes are mutually exclusive. The encryption attribute is also optional (defaults to true if not set).

The mode attribute can also be set to specify the connection mode. There are two supported connection modes: ssl and gsi. The ssl mode indicates a regular SSL connection mode. The gsi mode indicates a SSL connection mode with transport-level delegation support. The ssl mode is the default mode if the mode attribute is not specified. Please note that the gsi mode is intended for advanced users only.

Important! The credentials and certificate configuration is used only by the connector and is not used by the rest of the web services stack in Globus Toolkit. To configure credentials inside container please modify the Security Descriptor file.

Go to the <tomcat.dir>/webapps/<name>/WEB-INF/etc/globus_wsrf_core directory and modify content of global_security_descriptor.xml file setting paths to credential and key in <certitificate> section, where <name> is value of -Dwebapp.name property used during the deployment of wsjava into tomcat. If the property wasn't specified the default value is “wsrf”. For this guide tha <name> value was set to acgt.

```
<?xml version="1.0" encoding="UTF-8"?>
<containerSecurityConfig  xmlns="http://www.globus.org/security/descriptor/
container"   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.globus.org/security/descriptor
```

```
name_value_type.xsd"
xmlns:param="http://www.globus.org/security/descriptor" >
  <credential>
    <cert-key-files>
       <key-file value="/PATH/servicekey.pem"/>
           <cert-file value="/PATH/servicecert.pem"/>
        </cert-key-files>
  </credential>
</containerSecurityConfig>
```

- Add a HTTPS Valve in the <Engine name="Catalina" ... > section:

```
<Valve className="org.globus.tomcat.coyote.valves.HTTPSValve55"/>
```

You may have to edit <tomcat.dir>/webapps/<name>/WEB-INF/web.xml if you are running Tomcat on a non-default port, that is if not using port 8443 (HTTPS). For example, if you run Tomcat on port 443 using HTTPS then the WSRF servlet entry should be modified to have the following defaultProtocol and defaultPort parameters:

```
<web-app>
    ...
    <servlet>
        <servlet-name>WSRFServlet</servlet-name>
        <display-name>WSRF Container Servlet</display-name>
        <servlet-class>
            org.globus.wsrf.container.AxisServlet
        </servlet-class>
        <init-param>
            <param-name>defaultProtocol</param-name>
            <param-value>https</param-value>
        </init-param>
        <init-param>
            <param-name>defaultPort</param-name>
            <param-value>443</param-value>
        </init-param>
        <load-on-startup>true</load-on-startup>
    </servlet>
    ...
</web-app>
```

Alternatively, you can use the setDefaults Ant task to set the default protocol/port in the web.xml file:

```
$ cd $GLOBUS_LOCATION
$ ant -f share/globus_wsrf_common/tomcat/tomcat.xml setDefaults \
      -Dtomcat.dir=<tomcat.dir> \
      -DdefaultPort=<port> -DdefaultProtocol=<protocol>
```

Copy
$GLOBUS_LOCATION/lib/common/log4j-*.jar
and
$GLOBUS_LOCATION/lib/common/commons-logging-*.jar
files to <tomcat.dir>/webapps/<name>/WEB-INF/lib/ directory. Then configure the Log4j configuration file in <tomcat.dir>/webapps/<name>/WEB-INF/classes/ directory appropriately. The debugging settings will only affect the web application code.

Please always check the Tomcat log files under the <tomcat.dir>/logs directory for any errors or exceptions.

6.10. Starting Tomcat

It is recommended to increase the maximum heap size of the JVM when running the container. By default on Sun JVMs a 64MB maximum heap size is used. The maximum heap size can be set using the -Xmx JVM option. Please add the property -Xmx512M to JAVA_OPTS in the <tomcat>/bin/catalina.sh file.

IMPORTANT: By default Sun 1.4.x+ JVMs are configured to use /dev/random device as an entropy source. Sometimes the machine can run out of entropy and applications and using the /dev/random device will block until more entropy is available. One workaround for this issue is to configure the JVM to use /dev/urandom (non-blocking) device instead. For Sun JVMs a java.security.egd system property can be set to configure a different entropy source. Please add the -Djava.security.egd=file:/dev/urandom property to JAVA_OPTS in the <tomcat>/bin/catalina.sh file.

Start the tomcat using <tomcat-root>/bin/startup.sh script and check if it works listing available services. In any web browser try to open following page https://localhost: 8443/<name>/services, where <name> is value of -Dwebapp.name property used during the deployment of wsjava into tomcat. If the property wasn't specified the default value is “wsrf”.

The list of hosted services should be displayed:

```
And now... Some Services
• NotificationTestService (wsdl)
  • generateNotification
  • selfSubscribe
• TestAuthzService (wsdl)
  • addDeclinedMethod
  • SAMLRequest
• CounterService (wsdl)
  • add
  • createCounter
• TestServiceWrongWSDL (wsdl)
  • createResource
  • resetNumInstances
  • getInstanceInfo
  • testLocalInvocation
• ShutdownService (wsdl)
  • shutdown
• ...
```

There is some set of wsrf-core services, that are unnecessary to start simple gsi-enabled web service and can be removed. From the <tomcat-root>/webapps/<name>/WEB-INF/etc directory remove all files and directories except the “globus_wsrf_core” directory. If you want you can also remove all services except the gsi/AuthenticationService one from the <tomcat-root>/webapps/<name>/WEB-INF/etc/globus_wsrf_core/server-config.wsdd file.

The list of services should be:

```
And now... Some Services
• gsi/AuthenticationService (wsdl)
• requestSecurityTokenResponse
• requestSecurityToken
```

More details can be found in Apendix B "Implementation of simple Hello World server and client"

7. Grid Monitoring Portal.

For a dynamically changing Grid environment there is a need to provide tools for administrators for monitoring state of the infrastructure services.

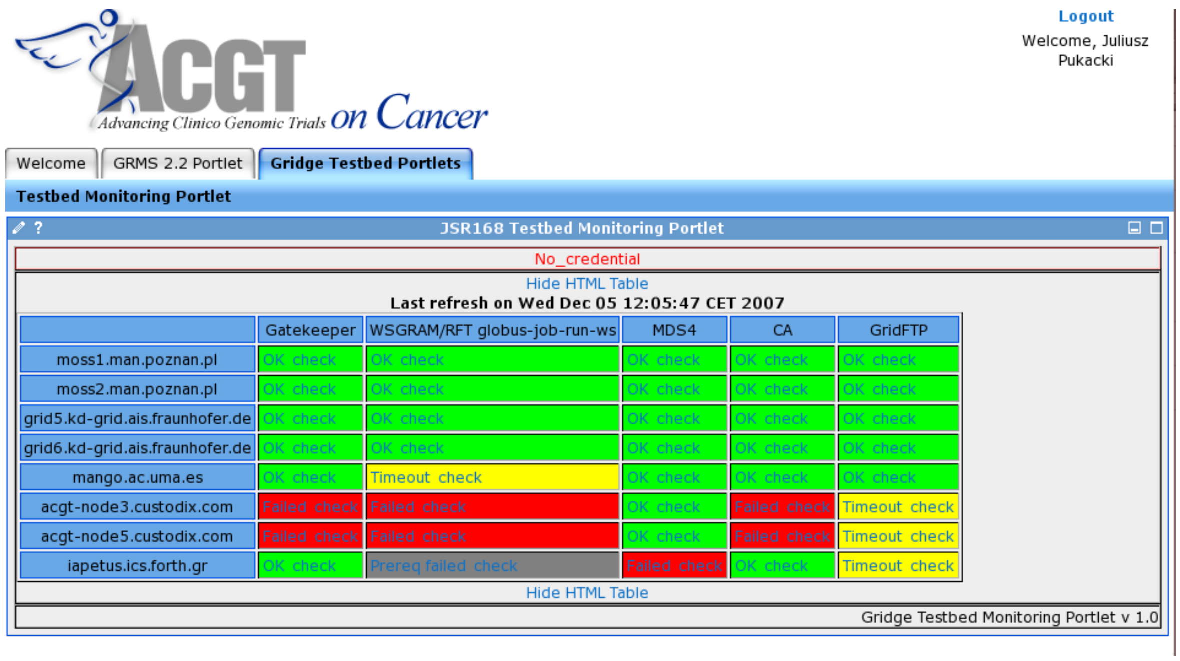
Grid Monitoring Portal is one of such a tool used for quick identification of services failures or other errors. It consists of two parts:

- testing tools
- publishing service

The testing tools are a framework for running parallel tests written in Java or any programming language through executing new processes, and a set of ready-to-go tests implemented mostly using Globus Java CoG Kit and shell scripts running native clients for different grid services. Results of these tests are written into a relational database. The framework can extended very easily to fit current needs of Grid Testbed configuration.

The publishing service is used for access to results of tests stored by testing tools in a relational database. Portlet is used as client for this service which displays current result set and history of tests. The web-client was extended to work with useful possibility of launching the test directly from the portlet, giving the administrator not only the tool for monitoring the testbed status, but also the ability to force the framework to start the desired test and see on-line the possibly changing test status/result.

In the table below the map the user can see a matrix of all hosts and services – so, it is easier to see the whole status and determine what is down and up. For every host we can also check the actual state of a service by clicking on the check hyperlink. (Fig. 4). A suitable colour tells if a service is up (green) or down (red), grey fields mean that on the given host there is no such service installed.


Logout
Welcome, Juliusz Pukacki
ACGT
Advancing Clinico Genomic Trials on Cancer
Welcome
GRMS 2.2 Portlet
Gridge Testbed Portlets
Testbed Monitoring Portlet
JSR168 Testbed Monitoring Portlet

No_credential

Hide HTML Table

Last refresh on Wed Dec 05 12:05:47 CET 2007

	Gatekeeper	WSGRAM/RFT globus-job-run-ws	MDS4	CA	GridFTP
moss1.man.poznan.pl	OK check	OK check	OK check	OK check	OK check
moss2.man.poznan.pl	OK check	OK check	OK check	OK check	OK check
grid5.kd-grid.ais.fraunhofer.de	OK check	OK check	OK check	OK check	OK check
grid6.kd-grid.ais.fraunhofer.de	OK check	OK check	OK check	OK check	OK check
mango.ac.uma.es	OK check	Timeout check	OK check	OK check	OK check
acgt-node3.custodix.com	Failed check	Failed check	OK check	Failed check	Timeout check
acgt-node5.custodix.com	Failed check	Failed check	OK check	Failed check	Timeout check
iapetus.ics.forth.gr	OK check	Prereq failed check	Failed check	OK check	Timeout check

Hide HTML Table

Gridge Testbed Monitoring Portlet v 1.0

Fig.4. Grid Monitoring portlet.

After clicking on the status value -OK or FAILED- another table is shown (), which displays a list of all conducted tests and their results. If a result is FAILED then the cause could be checked by clicking on the *failed* hyperlink (), then the reason is shown to the user.

The portlet is highly configurable – the user is able to add more services, hosts and centers which appear in the database, more human readable names can be set, details for administrator, web page, graphic files can be changed, depth of the tests' history can be set.

The list of implemented tests (only part of them are used in ACGT testbed):

- iStore - tests whether each machine is registered in iGrid central server called iStore. The test uses native iGrid client to get search.xml file and then processes it using Java XML tools to search for machines. If a machine is found, hardware info and list of jobmanagers are extracted.
- iServe - tests whether iGrid local part iServe (developed by GridLabWP-10) is running on each machine. The test runs gridlab-search-client against each machine. The test is implemented using native iGrid client.
- iGrid jobmans - tests whether any jobmanagers are listed in iStore for a given machine. The test is implemented in Java.
- merge jobmans - this test merges together lists of jobmanagers found in GRIS and in iStore. It is used as a prerequisite for other tests like Jobmanagers test. The test is implemented in Java.
- GRIS - tests whether each machine is running Grid Resource Information Service. Connects to each machine on port 2135, performs two unauthenticated LDAP queries with branch point Mds-Vo-name=local,o=grid. The first query is with scope ONELEVEL_SCOPE and filter (objectClass=*), getting information about CPU and OS. The second query is with scope SUBTREE_SCOPE and filter (objectClass=MdsServiceGram), searching for available jobmanagers. The test is implemented in Java using standard LDAP provider.
- Gatekeeper - tests whether Globus Gatekeeper is running on each machine. Tries to authenticate to a gatekeeper by calling Gram.ping() method. The test is implemented in Java using Globus Java CoG Kit 1.1 classes.
- mercury2 - tests whether the monitoring server version 2.X (developed by GridLabWP-11) is running on each machine. Runs the command monclient -n host.loadavg -p host=$MACHINE -v monp://$MACHINE for each machine. The test is implemented by using the native monitoring client.
- mercury2.3.1 - test whether monitoring software installed on each machine is version 2.3.1.
- GridFTP - tests whether each machine is running Grid File Transfer Protocol server. Connects to each machine to port 2811, performs GSI authentication to the Grid FTP server and then disconnects. The test is implemented in Java using Globus Java CoG Kit 1.1 classes.
- CA - tests whether each machine has certificates and policy files of all required Certification Authorities. Extracts all file names from CA tarball and submits a shell script to each remote machine to test existence of the same files in directory /etc/grid-security/certificates/. This test is run only if the Gatekeeper test was successful. The test is implemented using shell scripting and native globusrun client.
- Mapfile - tests whether each machine has required accounts in grid-mapfile. Gets all Distinguished Names (DN) from master grid-mapfile, reads grid-mapfile from each machine by submitting /bin/cat /etc/grid-security/grid-mapfile, extracts DNs from it and

compares them to DNs from master grid-mapfile. This test is run only if Gatekeeper test was successful. The test is implemented using shell scripting and native globusrun client.

- GSISSH - tests whether each machine is running Secure Shell (SSH) daemon with Grid Security Infrastructure (GSI) extensions. Runs command gsissh -p 2222 $MACHINE echo "TESTSTRING" and searches for "TESTSTRING" in the output. The test is implemented by calling native gsissh client.
- Software - tests whether the required software is installed. Submits a complex shell script to each remote machine, which first sources /etc/gridlab.conf file to set the environment variables and then tries to find the required software packages. This test is run only if the Gatekeeper test was successful. The test is implemented using shell scripting and native globusrun client.
- Java - tests whether java implementation on the remote machine is at least version 1.3. This test submits a remote shell script using native globusrun client.
- Delphoi - tests whether GridLab adaptive software is running on each machine. Runs command control-pythia check -host $MACHINE for each machine. The test is implemented using native adaptive client.
- Jobmanagers - tests jobmanagers as advertised by GRIS or iGrid on each machine. Gets all jobmanagers found by the "merge jobmans" test and submits job /bin/echo "XXX TEST XXX" to each one. Searches for XXX TEST XXX in the job outbut. So the number of tested jobmanagers depends on result of GRIS and iStore test. This is a "composite" test which produces variable number of results. This test is run only if Gatekeeper test was succesful and at least one jobmanager is available. The test is implemented in Java using Globus Java CoG Kit 1.1 classes.
- GRMS - tests whether Grid Resource Management Service can submit a job to each jobmanager. Calls operation submitJob(String xml,StringHolder jobId) on the GRMS service to submit /bin/date as a simple job to each jobmanager. Then it polls the GRMS for the job status until it finishes. The test is implemented in Java using Apache Axis and Globus 3 classes.
- Mpicc - tests whether it is possible to compile C programs using Message Passing Interface libraries. Submits a complex shell script, which downloads source of a simple MPI C program (cpi.c) using globus-url-copy from GASS server, then compiles the source. Tries to cope with many different operating systems, but may fail for unknown OSes. This test is run only if the Gatekeeper test was successful. The test is implemented using shell scripting and native globusrun client.
- mpif77 - tests whether it is possible to compile FORTRAN77 programs using Message Passing Interface libraries. Submits a complex shell script, which downloads the source of a simple MPI F77 program (mpi_prog.f) using globus-url-copy from GASS server, then compiles the source. Tries to cope with many different operating systems, but may fail for unknown OSes. This test is run only if the Gatekeeper test was successful. The test is implemented using shell scripting and native globusrun client.
- MPI-C - tests whether MPI C program can be run on all jobmanagers advertised by GRIS on each machine. Gets all jobmanagers found by the GRIS test and submits an MPI job which runs the MPI C program previously compiled by the mpicc test. The RSL for the job includes (jobType=mpi)(count=2) to run the job on two CPUs. Currently ignores Condor jobmanagers, as they don't support MPI jobs. This is a "composite" test which produces a variable number of results. This test is run only if the Gatekeeper test was successful and GRIS reported that at least one jobmanager and mpicc test was succesful. The test is implemented in Java using Globus Java CoG Kit 1.1 classes.

- MPI-f77 - tests whether MPI FORTRAN77 program can be run on all jobmanagers advertised by GRIS on each machine. Gets all jobmanagers found by the GRIS test and submits an MPI job which runs the MPI F77 program previously compiled by the mpif77 test. The RSL for the job has (jobType=mpi)(count=2) to run the job on two CPUs. Currently ignores Condor jobmanagers, as they don't support MPI jobs. This is a "composite" test which produces variable number of results. This test is run only if the Gatekeeper test was successful and GRIS reported that at least one jobmanager and mpif77 test was successful. The test is implemented in Java using Globus Java CoG Kit 1.1 classes.
- GRMS MPI - test whether GridLab Resource Management Service developed by WP-9 can submit an MPI job to each jobmanager. Calls the operation submitJob(String xml,StringHolder jobId) on the GRMS service to a submit cpi executable compiled during the "mpicc" test as an MPI job for two CPUs to each jobmanager found in the GRIS test. Then, it polls the GRMS for the job status until it finishes. Some jobmangers are disabled because they are known not to handle MPI jobs. The test is implemented in Java using Apache Axis and Globus 3 classes.
- GAT - gets the sources of GridLab GAT engine, compiles them on each machine and runs their tests by running the command make -C engine tests. The test depends heavily on correct settings in /etc/gridlab.conf.
- Mercury compile - gets the latest source of Mercury monitoring tool and tries to compile it on each machine. The test depends heavily on correct settings in /etc/gridlab.conf. Problems should be consulted with GridLab WP-11 (Monitoring).
- Triana – gets Triana from CVS and tries to run it on remote machines.
- Cactus – gets Cactus with C-GAT thorns sources from CVS and tries to compile them against GAT and adaptors on the remote machine.
- GRMS WS – tests if the GRMS Web Service is installed on given machine. Simple call of common getServiceDescription() method is performed during test.
- GAS – tests if the GAS Web Service is installed on given machine. Simple call of common getServiceDescription() method is performed during test.
- eNanos – tests if the eNanos web service is installed and if it is working – it uses a simple command ping for the target service
- UNICORE – checks if target unicore system is available – it checks if target unicore gateway is on-line and then checks if other unicore hosts are available and belong to the given VO unicore site
- JOSH – checks if target host is up, if service is installed and running, then tries to submit a very simple job

There is also other portlet within Monitoring Portal. It shows a map of Europe and all the machines participating the testbed. It uses data gathered in the database and renders it so that users can see the actual status of the whole portlet.

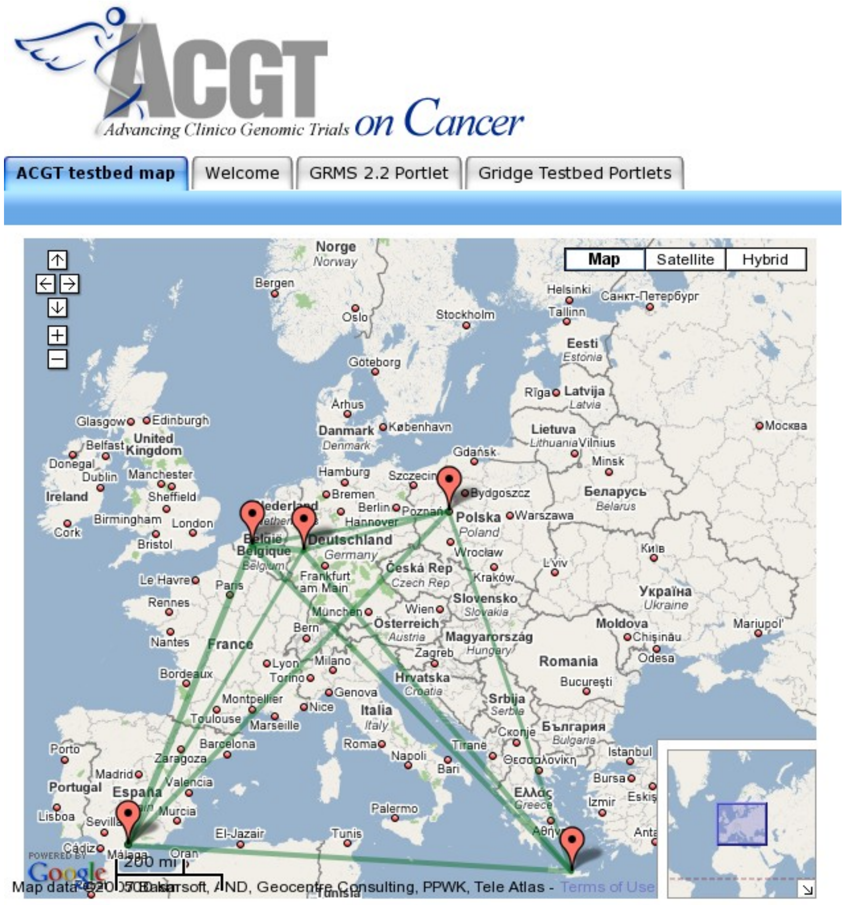

ACGT
Advancing Clinico Genomic Trials on Cancer
ACGT testbed map
Welcome
GRMS 2.2 Portlet
Gridge Testbed Portlets
Map
Satellite
Hybrid
200 mi
Map data ©2007 Basarsoft, AND, Geocentre Consulting, PPWK, Tele Atlas - Terms of Use

Fig.5. Grid Testbed Map

References

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

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

[3] "GRMS Admin Guide"
http://www.gridge.org/files/grms/doc/admin/pdf/view/GrmsAdminGuide.pdf

[4] "GRMS User Guide"
http://www.gridge.org/files/grms/doc/user/pdf/view/GrmsUserGuide.pdf

[5] "GDMS Admin Guide"
http://www.gridge.org/files/dms/doc/admin/pdf/view/DMSAdminGuide.pdf

[6] "GDMS User Guide"

http://www.gridge.org/files/dms/doc/user/pdf/view/DMSUserGuide.pdf

[7] Russell Butek "Which style of WSDL should I use?" http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/

[8] "Axis Reference Guide" http://ws.apache.org/axis/java/reference.html

[9] "Authorization Framework Documentation"
http://www.globus.org/toolkit/docs/development/4.1.2/security/authzframe/index.html

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

Appendix B: Implementation of simple Hello World server and client

This chapter shows how to implement simple gsi-enabled web service and client. The service has only one method hello with no arguments and it returns depending on version information about the caller (the user Distinguished Name) or complex information about proxy delegated to the service.

IMPORTANT All classes and jars containing examples presented below were compiled using java 1.5.

1. Creating WSDL file describing the service interface

This point can be omitted if you want to build the client or service basing on the existing WSDL file.

Interface of every service must be described in Web Service Definition Language. The WSDL document can be written manually or generated using appropriate tools. One of such tools that can be used to generate WSLD file is Java2WSDL tool provided with Axis.

- Design the service interface as java class making public all methods you want to be available. IMPORTANT: Do not use java interface to model the service. In such case names of parameters in generated WSDL file will be arg1, arg2 ... argN instead of real ones.

```
package acgt.examples.hello;
public class HelloException extends Throwable {
   public int errorCode;
   public String errorMessage;
}
ServiceException.java
package acgt.examples.hello;
public class Hello {
  public String hello() throws HelloException  {
     return null;
  }
}
```

Hello.java

- Compile the class modeling the service interface with option switching on additional debug information (“-g” option of javac tool). Otherwise the java2WSDL tool will not be able to obtain real names of parameters.

javac -g *.java

- Use the java2WSDL tool to generate the WSDL file

Usage: java org.apache.axis.wsdl.Java2WSDL [options] class-of-portType

Detailed information about the java2WSDL tool can be found on Axis Reference Guide page.

The most important options are:

-o, --output <argument>

```
  output WSDL filename
-l, --location <argument>
  service location url
-P, --portTypeName <argument>
  portType name (obtained from class-of-portType if not specified)
-b, --bindingName <argument>
  binding name (--servicePortName value + "SOAPBinding" if not specified)
-S, --serviceElementName <argument>
  service element name (defaults to --servicePortName value + "Service")
-s, --servicePortName <argument>
  service port name (obtained from --location if not specified)
-n, --namespace <argument>
  target namespace
-p, --PkgtoNS <argument>=<value>
  package=namespace, name value pairs
-N, --namespaceImpl <argument>
  target namespace for implementation wsdl
-A, --soapAction <argument>
  value of the operations soapAction field. Values are DEFAULT,
  OPERATION or NONE. OPERATION forces soapAction to the name
  of the operation. DEFAULT causes the soapAction to be set
  according to the operations meta data (usually ""). NONE forces
  the soapAction to "". The default is DEFAULT.
-y, --style <argument>
  The style of binding in the WSDL, either DOCUMENT, RPC, or WRAPPED.
-u, --use <argument>
  The use of items in the binding, either LITERAL or ENCODED
```

Following set of options is recommend:

```
 --soapAction OPERATION
 --style WRAPPED
 --use LITERAL
```

The“--location” option is mandatory.

The "Which style of WSDL should I use?" [7] article describes differences between different styles of WSDL files.

```
java -cp
.:./jars/axis.jar:./jars/jaxrpc.jar:./jars/log4j-1.2.13.jar:./jars/commons-
logging-1.1.jar:./jars/commons-discovery-0.2.jar:./jars/wsdl4j-1.5.1.jar:./
jars/saaj.jar \
   org.apache.axis.wsdl.Java2WSDL \
      --style WRAPPED \
      --use LITERAL \
      --soapAction OPERATION \
      --namespace http://www.eu-acgt.org/ \
      --namespaceImpl http://www.eu-acgt.org/ \
      --location HelloLocation \
      --output hello.wsdl \
   acgt.Hello
```

All the jars needed for Java2WSDL tool can be taken from GT4.1.2 or downloaded as a separated tarball.

As a result the Java2WSDL tool should generate the hello.wsdl file.

```
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="http://www.eu-
acgt.org/" xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="hello">
    <complexType/>
   </element>
   <element name="helloResponse">
    <complexType>
     <sequence>
      <element name="helloReturn" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <complexType name="HelloException">
    <sequence>
     <element name="errorCode" type="xsd:int"/>
     <element name="errorMessage" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="fault" type="impl:HelloException"/>
  </schema>
 </wsdl:types>

   <wsdl:message name="helloRequest">
      <wsdl:part element="impl:hello" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="HelloException">
      <wsdl:part element="impl:fault" name="fault"/>
   </wsdl:message>
   <wsdl:message name="helloResponse">
      <wsdl:part element="impl:helloResponse" name="parameters"/>
   </wsdl:message>

   <wsdl:portType name="Hello">
      <wsdl:operation name="hello">
         <wsdl:input message="impl:helloRequest" name="helloRequest"/>
         <wsdl:output message="impl:helloResponse" name="helloResponse"/>
         <wsdl:fault message="impl:HelloException" name="HelloException"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="HelloSoapBinding" type="impl:Hello">
      <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="hello">
         <wsdlsoap:operation soapAction="http://www.eu-acgt.org/hello"/>
         <wsdl:input name="helloRequest">
```

```
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="helloResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
         <wsdl:fault name="HelloException">
            <wsdlsoap:fault name="HelloException" use="literal"/>
         </wsdl:fault>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="HelloService">
      <wsdl:port binding="impl:HelloSoapBinding" name="Hello">
         <wsdlsoap:address location="HelloLocation"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Please modify the value of soapAction in binding section to be in url format, otherwise invoking the client you will get following error:

ERROR handler.AddressingHandler [main,invoke:120] Exception in AddressingHandler

```
org.apache.axis.types.URI$MalformedURIException: No scheme found in URI.
        at org.apache.axis.types.URI.initialize(URI.java:653)
```

In the example value of soapAction was changed to:

```
<wsdlsoap:operation soapAction="http://www.eu-acgt.org/hello"/>
```

2. Implementing gsi-enabled Web Service

2.2. Generating stub classes

Having the WSDL file describing the service's interface, use the WSDL2java tool to generate auxiliary axis classes hiding the complexity of SOAP communication.

Usage: java org.apache.axis.wsdl.WSDL2Java [options] WSDL-URI

Detailed information about the WSDL2java tool can be found on Axis Reference Guide page.

The most important options are:

-s, --server-side

   emit server-side bindings for web service

-S, --skeletonDeploy <argument>

   deploy skeleton (true) or implementation (false) in deploy.wsdd.

   Default is false. Assumes --server-side.

-N, --NStoPkg <argument>=<value>

   mapping of namespace to package

-f, --fileNStoPkg <argument>

   file of NStoPkg mappings (default NStoPkg.properties)

-p, --package <argument>

override all namespace to package mappings, use this package name instead

-o, --output <argument>

output directory for emitted files

-u, --allowInvalidURL

emit file even if WSDL endpoint URL is not a valid URL

-B, --buildFile

emit Ant Buildfile for web service

Use the --server-side option if you want the auxiliary code for service side to be created. Otherwise the tool will generate only client side code. It is also recommended to use the --skeletonDeploy option set to “true”, what makes thedeploy.wsd file, describing the service during the deployment simpler.

```
java -cp .:./jars/axis.jar:./jars/log4j-1.2.13.jar:./jars/commons-logging-1.1.jar:./jars/commons-discovery-0.2.jar:./jars/jaxrpc.jar:./jars/saaj.jar:./jars/wsdl4j-1.5.1.jar \
    org.apache.axis.wsdl.WSDL2Java \
      --server-side \
      --skeletonDeploy true \
      --package acgt.examples.hello.stub \
      --allowInvalidURL \
      --buildFile \
      --output wsdl2java \
    hello.wsdl
```

All jars needed for WSDL2Java tool can be taken from GT4.1.2 or downloaded as a separated tarball.

Following set of files should be generated:

```
./wsdl2java
|-- acgt
|   `-- examples
|       `-- hello
|           `-- stub
|               |-- HelloException.java
|               |-- HelloService.java
|               |-- HelloServiceLocator.java
|               |-- HelloSoapBindingImpl.java
|               |-- HelloSoapBindingSkeleton.java
|               |-- HelloSoapBindingStub.java
|               |-- Hello_PortType.java
|               |-- deploy.wsdd
|               `-- undeploy.wsdd
`-- build.xml
```

2.2. Implementing the service functionality

Previous step (WSDL2java) should generate set of java classes including the Hello_PortType class containing definition of service's portType (interface) and the HelloSoap_BindingImpl class one implementing this portType/interface.

```
/**
 * Hello_PortType.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.4 Mar 01, 2007 (10:42:15 CST) WSDL2Java emitter.
 */
package acgt.examples.hello.stub;
public interface Hello_PortType extends java.rmi.Remote {
    public java.lang.String hello() throws java.rmi.RemoteException,
acgt.examples.hello.stub.HelloException;
}
/**
 * HelloSoapBindingImpl.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.4 Mar 01, 2007 (10:42:15 CST) WSDL2Java emitter.
 */
package acgt.examples.hello.stub;

public class HelloSoapBindingImpl implements
acgt.examples.hello.stub.Hello_PortType{
    public java.lang.String hello() throws java.rmi.RemoteException,
acgt.examples.hello.stub.HelloException {
        return null;
    }
}
```

Please, fill in interface methods in HelloSoapBindingImpl.java file with real service functionality.

2.2.1. Hello World!: Showing who's calling

The service returns information about indentity (subject of certificate aka. Distinguished Name) of the user who invoked the hello method.

```
/**
 * HelloSoapBindingImpl.java
 */

package acgt.example.hello.stub;

import java.security.Principal;
import javax.security.auth.Subject;
import org.apache.axis.MessageContext;
import org.globus.wsrf.impl.security.authentication.Constants;

public class HelloSoapBindingImpl implements
acgt.examples.service.stub.Hello_PortType
{
    public java.lang.String hello() throws java.rmi.RemoteException,
acgt.examples.hello.stub.HelloException
    {
        String userDN = null;
        Subject subject =
(Subject)MessageContext.getCurrentContext().getProperty(Constants.PEER_SUBJ
ECT);

        if( subject != null)
        {
           Principal principal = null;
           if( subject.getPrincipals().isEmpty() == false)
           {
              principal =
(Principal)subject.getPrincipals().iterator().next();
              if( principal != null)
              {
                 userDN = principal.getName();
              }
           }
        }
        return userDN;
    }
}
```

Following set of jars is needed to compile the code:

axis.jar

jaxrpc.jar

wsrf_core.jar

All these jars can be taken from Globus Toolkit 4.1.2 or downloaded as a separated tarball.

Compile all classes generated by WSDL2Java tool including the modified HelloSoapBindingImpl.java one and generate jar archive.

To generate the hello.jar yourself download the tarball (containing ant build file, all needed jars and code) file and simply invoke ant tool:

ant

The Apache Ant is required!

2.2.2.Hello World!: Showing information about delegated proxy

The service returns full information about proxy certificate that was delagated to the service (Distinguished Name, proxy type, proxy lifetime, etc.).

```
/**
 * HelloSoapBindingImpl.java
 */
package acgt.examples.hello.stub;
import java.security.Principal;
import javax.security.auth.Subject;
import javax.security.auth.Subject;
import org.apache.axis.MessageContext;
import org.ietf.jgss.GSSCredential;
import org.globus.wsrf.impl.security.authentication.Constants;
import org.globus.gsi.GlobusCredential;
import org.globus.gsi.gssapi.GlobusGSSCredentialImpl;

public class HelloSoapBindingImpl implements
acgt.examples.hello.stub.Hello_PortType {
    public java.lang.String hello() throws java.rmi.RemoteException,
acgt.examples.hello.stub.HelloException {
        GSSCredential credential = null;
        Subject subject =
(Subject)MessageContext.getCurrentContext().getProperty(Constants.PEER_SUBJ
ECT);
        if( subject != null)
        {
            try
            {
                if( subject.getPrivateCredentials().isEmpty() == false)
                {
                    credential =
(GSSCredential)subject.getPrivateCredentials().iterator().next();
                    if (credential != null && credential instanceof
GlobusGSSCredentialImpl) {
                        GlobusCredential globusCred =
((GlobusGSSCredentialImpl)credential).getGlobusCredential();

                        return globusCred.toString();
                    }
                    else
                        return "No Globus proxy";
                }
            }
            catch( Exception e){
                return "Failed to get user's credential: " + e;
            }
        }
        return "ERROR";
    }
}
```

Following set of jars is needed to compile the code:

axis.jar

jaxrpc.jar

wsrf_core.jar

cog-jglobus.jar

All these jars can be taken from Globus Toolkit 4.1.2 or downloaded as a separated tarball.

Compile all classes generated by WSDL2Java tool including the modified HelloSoapBindingImpl.java one and generate jar archive.

To generate the hello.jar yourself download the tarball (containing ant build file, all needed jars and code) file and simply invoke ant tool:

ant

The Apache Ant is required!

2.3. Deploying Service

- copy jar with code implementing the service's functionality to the <tomcat>/webapps/<name>/WEB-INF/lib directory
- create a directory in <tomcat>/webapps/<name>/WEB-INF/etc

- copy deploy.wsdd file generated by WSDL2Java tool to the created directory under the server-config.wsdd name.

After tomcat restart (<tomcat>/bin/shutdown.sh and <tomcat>/bin/startup.sh) the deployed service should be listed on list of deployed service. Using any Web browser check the list of services inter the following location http://localhost:8080/<name>/services, where name is name choosen during the deploying globus to the tomcat. If the name wasn't specified the default one is "wsrf".

And now... Some Services

* Hello (wsdl)
    o hello
* gsi/AuthenticationService (wsdl)
    o requestSecurityTokenResponse
    o requestSecurityToken

3. Implementing gsi-enabled Web Service Client

The aim of this part of guide is to show step by step how to write and configure simple gsi-enabled command line java client, able to contact service described in previous chapter without and with delegation of user proxy certificate.

3.1. Generating stub classes

Having the WSDL file describing the service's interface, use the WSDL2java tool to generate auxiliary axis classes hiding the complexity of SOAP communication.You can skip this point and use stub classes generated for the service or generate them once again limiting the process only to classes needed on client side.

Usage: java org.apache.axis.wsdl.WSDL2Java [options] WSDL-URI

Detailed information about the WSDL2java tool can be found on Axis Reference Guide page.

The most important options are:

```
-N, --NStoPkg <argument>=<value>
   mapping of namespace to package
-f, --fileNStoPkg <argument>
   file of NStoPkg mappings (default NStoPkg.properties)
-p, --package <argument>
   override all namespace to package mappings, use this package name instead
-o, --output <argument>
   output directory for emitted files
-u, --allowInvalidURL
   emit file even if WSDL endpoint URL is not a valid URL
-B, --buildFile
   emit Ant Buildfile for web service
```

Do not use the --server-side option if you want only the client side code to be generated.

```
java -cp .:./jars/axis.jar:./jars/log4j-1.2.13.jar:./jars/commons-logging-1.1.jar:./jars/commons-discovery-0.2.jar:./jars/jaxrpc.jar:./jars/saaj.jar:./jars/wsdl4j-1.5.1.jar \
   org.apache.axis.wsdl.WSDL2Java \
     --package acgt.examples.hello.stub \
     --allowInvalidURL \
     --buildFile \
     --output wsdl2java \
   hello.wsdl
```

All jars needed for WSDL2Java tool can be taken from GT4.1.2 or downloaded as a separated tarball.

Following set of files should be generated:

```
./wsdl2java
|-- acgt
|   `-- examples
|       `-- hello
|           `-- stub
|               |-- HelloException.java
|               |-- HelloService.java
|               |-- HelloServiceLocator.java
|               |-- HelloSoapBindingStub.java
|               `-- Hello_PortType.java
`-- build.xml
```

3.2. Client code

The client code is relatively simple, but its more important parts were marked by numbers in comments and explained below.

```
package acgt.examples.hello;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import acgt.examples.hello.stub.Hello_PortType;
import acgt.examples.hello.stub.HelloServiceLocator;
import acgt.examples.hello.stub.HelloException;
import org.apache.axis.client.Stub;
import org.globus.axis.gsi.GSIConstants;
import org.globus.axis.util.Util;
import org.globus.wsrf.impl.security.authentication.Constants;
import org.globus.wsrf.impl.security.authorization.IdentityAuthorization;
import org.gridforum.jgss.ExtendedGSSManager;
import org.ietf.jgss.GSSCredential;
import org.ietf.jgss.GSSException;

public class HelloClient
{
   public static void main( String[] args)
   {
      String url = args[0];
      URL hello_url = null;
      try
      {
         hello_url = new URL( url);
      }
      catch( MalformedURLException e1)
      {
         System.err.println( "Malformed url: " + url);
         System.exit( 1);
      }
      String hello_dn = args[1];
      ExtendedGSSManager manager =
(ExtendedGSSManager)ExtendedGSSManager.getInstance();
      GSSCredential credential = null;
      try
      {
/* 1 */  credential =
manager.createCredential( GSSCredential.INITIATE_AND_ACCEPT);
      }
      catch( GSSException e)
      {
         System.err.println( "Failed to load user's proxy");
         System.exit( 1);
      }
      HelloServiceLocator locator = new HelloServiceLocator();
      Hello_PortType hello = null;
      try
      {
/* 2 */  hello = locator.getHello( hello_url);
      }
      catch( ServiceException e)
      {
         System.err.println( "Failed to get port type");
         System.exit( 1);
      }
/* 3 */  ((Stub)hello)._setProperty( GSIConstants.GSI_CREDENTIALS,
credential);
/* 4 */  ((Stub)hello)._setProperty( GSIConstants.GSI_MODE,
GSIConstants.GSI_MODE_FULL_DELEG);
/* 5 */  ((Stub)hello)._setProperty( Constants.GSI_TRANSPORT,
Constants.ENCRYPTION);
/* 6 */  ((Stub)hello)._setProperty( Constants.GSI_SEC_CONV,
Constants.SIGNATURE);
/* 7 */  ((Stub)hello)._setProperty( Constants.AUTHORIZATION, new
IdentityAuthorization( hello_dn));
/* 8 */  Util.registerTransport();
      try
      {
/* 9 */  String user = hello.hello();
         System.out.println( user);
      }
      catch( HelloException e)
      {
         System.err.println( "Operation failed:");
         System.err.println( "errorCode: " + e.getErrorCode());
         System.err.println( "errorMessage: " + e.getErrorMessage());
      }
      catch( RemoteException e)
      {
         System.err.println( e);
      }
   }
}
```

Marked parts have following meanings:

(1) Loads user credential from default location. Please see "Client configuration" section for details,

(2) Creates and returns object representing Hello interface,

(3) Sets user certificate to be used to authenticate the user,

(4) Sets delegation mode to "full". If you do not want to delegate the proxy replace GSIConstants.GSI_MODE_FULL_DELEG constatnt with GSIConstants.GSI_MODE_NO_DELEG one

(5) Forces client to encrypt communication, that guaranties as well privacy as integrity,

(6) Tells the client to use only digital signatures to protect messages integrity for "Secure Conversation", encryption is not needed on this level because the whole "channel" is encrypted,

(7) Sets service Distinguished Name that will be used during the mutual authentication, must be used if delegation of certificate was chosen.Delegation of proxy certificate to unknown service is not allowed,

(8) Registers axis transports for https and httpg protocols,

(9) Invokes remote method.

3.3. Client configuration

To work properly, client has to be able to load user's proxy certificate and validate service credential during the handshake procedure. To do this it needs to know location of the file containing proxy certificate and directory containing public keys of Certificate Authorities it should trust.

- Client looks for proxy according to following ruls:

It first checks the X509_USER_PROXY system property. If the property
is not set, it checks next the 'proxy' property in the current
configuration. If that property is not set, then it defaults to a
value based on the following rules:
If a UID system property is set, and running on a Unix machine it
returns /tmp/x509up_u${UID}. If any other machine then Unix, it returns
${tempdir}/x509up_u${UID}, where tempdir is a platform-specific
temporary directory as indicated by the java.io.tmpdir system property.
If a UID system property is not set, the username will be used instead
of the UID. That is, it returns ${tempdir}/x509up_u_${username}

- Client looks for the CA directory according to following rules:

It first checks the X509_CERT_DIR system property. If the property
is not set, it checks next the 'cacert' property in the current
configuration. If that property is not set, it tries to find
the certificates using the following rules:
First the ${user.home}/.globus/certificates directory is checked.
If the directory does not exist, and on a Unix machine, the

/etc/grid-security/certificates directory is checked next.
If that directory does not exist and GLOBUS_LOCATION
system property is set then the ${GLOBUS_LOCATION}/share/certificates
directory is checked.

CoG library configuration can be modified using the COG properties file ~/.globus/cog.properties

```
#Java CoG Kit Configuration File
proxy=/tmp/x509up_u501
cacert=/etc/grid-security/certificates/
```

3.4. Running the client

Following set of jars is needed to run the client:

Globus jars (download tarball):

addressing-1.0.jar
axis.jar
cog-axis.jar
cog-jglobus.jar
commons-codec-1.3.jar
commons-discovery-0.2.jar
commons-httpclient-3.0.jar
commons-logging-1.1.jar
cryptix-asn1.jar
cryptix.jar
cryptix32.jar
globus_java_authz_framework.jar
jaxrpc.jar
jce-jdk13-131.jar
log4j-1.2.13.jar
opensaml-1.1.jar
puretls.jar
saaj.jar
wsdl4j-1.5.1.jar
wsrf_core.jar
wsrf_core_stubs.jar
wsrf_provider_jce.jar
wss4j.jar
xalan-2.6.jar

xmlsec-1.2.1.jar

Additionally the client-config.wsdd file from Globus 4.1.2 is has to be placed in directory listed on CLASSPATH.

IMPORTANT: By default Sun 1.4.x+ JVMs are configured to use /dev/random device as an entropy source. Sometimes the machine can run out of entropy and applications and using the /dev/random device will block until more entropy is available. One workaround for this issue is to configure the JVM to use /dev/urandom (non-blocking) device instead. For Sun JVMs a java.security.egd system property can be set to configure a different entropy source.

```
-Djava.security.egd=file:/dev/urandom
```

Client usage:

```
java -Djava.security.egd=file:///dev/urandom
acgt.examples.hello.HelloClient <SERVICE_URL> <SERVICE_DN>
```

for example:

```
#!/bin/bash
CLASSPATH=.
for i in ./jars/*.jar; do CLASSPATH=$CLASSPATH:$i; done
java -cp $CLASSPATH -Djava.security.egd=file:///dev/urandom \
            acgt.examples.hello.HelloClient \
            https://localhost:8443/HELLO/services/Hello \
            "/C=PL/O=GRID/O=PSNC/CN=Hello/druid-bis.man.poznan.pl"
```

Following results should be diplayed:

- for Hello World!: Showing who's calling

```
/C=PL/O=GRID/O=PSNC/CN=Tomasz Piontek
```

- for Hello World!: Showing information about delegated proxy

```
subject     : C=PL,O=GRID,O=PSNC,CN=Tomasz
Piontek,CN=1168110104,CN=2070173049
issuer      : C=PL,O=GRID,O=PSNC,CN=Tomasz Piontek,CN=1168110104
strength    : 512 bits
timeleft    : 31931 sec
proxy type  : Proxy draft compliant impersonation proxy
```