Table of Contents

1 WMProxy Service

1.1 About WMProxy

WMProxy is a new component to access the **EMI Workload Management System (WMS)**.

The need of such a component, besides adhering to the emerging design methodology called **Service Oriented Architecture (SOA)**, is to improve the performance of existing similar components, to handle more efficiently a large number of requests for job submission and control and also to provide additional features.

The WMProxy is implemented as a *Web service*. The Web is considered a successful technology mostly because of its simplicity and ubiquity. A Web service allows us to take advantage of the benefits of the Web, not only to provide information, but also to offer services to a greater community of possible users.

Since the **European Middleware Initiative (EMI)** Middleware is a composition of Grid services provided by different vendors and/or operated by different organizations, the Web services technology is a powerful means to achieve service interoperability and allow easier compliance with emerging standards such as **Open Grid Services Architecture (OGSA)** and **Web Services Resource Framework (WSRF)**.

1.2 Client configuration

C++ WMProxy client commands behavior may be configured by editing a proper configuration file. Here follow the steps needed to properly configure file location on client side:

1. For each supported Virtual Organisation a directory must be created, with the following name:

   ```
   /etc/glite-wms/<vo name-lowercase>
   (e.g. for dteam /etc/glite-wms/dteam)
   ```
2. Files installation:
   org.glite.wms.client (version **> 3.1.9**) installs a template file in the following location:

   ```
   /etc/vo_template/glite_wmsui.conf.template
   ```

   This file should be copied with the name "glite_wmsui.conf" in each directory created in **step 1)**

   for backward compatibility (version **<= 3.1.9**) old configuration file name approach is still accepted

   ```
   /etc/vo_template/glite_wms.conf
   ```

   This file should be copied with the name "glite_wms.conf" in each directory created in **step 1)**
3. Edit the configuration files, e.g.:
   /etc/glite-wms/dteam/glite_wms.conf
   and check if desired parameters are correctly set, e.g.:

   ```
   [
           WMProxyEndPoints = {"https://ghemon.cnaf.infn.it:7443/glite_wms_wmproxy_server"};
           ErrorStorage = "/var/tmp";
           OutputStorage= "/tmp";
           ListenerStorage = "/tmp";
           VirtualOrganisation = "dteam";
           JdlDefaultAttributes=[
                   RetryCount = 3;
                   ShallowRetryCount = 3;
                   rank =  -other.GlueCEStateEstimatedResponseTime;
                   requirements = other.GlueCEStateStatus == "Production" ;
                   myCustomizedAttribute = "foo";
   ```

```
                SignificantAttributes={"rank","requirements","fuzzyrank"};
                ]
        ]
```

For backward compatibility (version **<= 3.1.9**), `WmsClient` section of the configuration file will contain above attributes

4. Configuration file attributes description
   A brief summary of all supported attributes for the configuration file follows.
   `WMProxyEndPoints`: list of endpoints URL of the WMProxy to contact.
   `ErrorStorage`: path of the directory where the UI creates log files.
   `OutputStorage`: path of the directory where the job OutputSandbox files are stored if not specified by the user through commands options.
   `ListenerStorage`: path of the directory where are created the pipes where the glite_wms_console_shadow process saves the job standard streams for interactive jobs.
   `VirtualOrganisation`: the virtual organisation used to perform the operation
   `JdlDefaultAttributes`: (version >= 3.1.13) This is a classad attribute and allows the user specifying a set of attributes/values that will be automatically inserted in the submitting JDL
   For backward compatibility (since version < 3.1.13, `JdlDefaultAttributes` section not present) following attriubtes will be taken in consideration:
   `Rank`: default value for the ranking expression in the JDL.
   `Requirements`: default value for the requirements expression in the JDL (a further condition on the submitting requirements will be added)
   `RetryCount`: default value for the RetryCount attribute in the JDL.
   `MyProxyServer`: MyProxy server address. To be set only if proxy renewal has to be enabled for all requests.
   `LBAddress`: this attribute forces the WMProxy server to register jobs and/or log events to the specified LB.
5. Configuration File Priority
   Attributes listed configuration file described in **step 2)** may be overridden by (in order of priority):

   `--config` option of WMProxy Client commands
   $HOME/.glite/<vo name>/glite_wmsclient.conf
   $GLITE_WMS_CLIENT_CONFIG

   Please note that the attributes not present in the customized file, yet listed in file described in **step 2)**, are inherited by final configuration attributes. If the user does not want to inherit a specific attribute, he may assign a null/empty value (depending on the type).
   Please notice that default configuration files will be parsed anyway: default attributes will be taken in consideration when not found on customized configuration file.
6. Other client configuration steps:
   - Make sure `$GLOBUS_LOCATION/lib` and `$GLITE_LOCATION/lib` are included in the `LD_LIBRARY_PATH`
   - Add `$GLITE_LOCATION/bin` to your `PATH`

1.3 Architecture

To adhere to the SOA model, WMProxy has been designed and implemented as a **Simple Object Access Protocol (SOAP)** Web service. The interface is described through the **Web Service Description Language (WSDL)**. The WSDL file was written following the **Web Services Interoperability Basic Profile (WS-I Basic Profile)**. This profile defines a set of Web Services specifications that promote interoperability. The WMProxy service runs in an Apache container extended with **Fast CGI** and **Grid Site** modules.

- The **Fast CGI** module provides Common Gateway Interface (CGI) functionality with some other specific features. The most important advantage of Fast CGI is its performance and persistence. Fast

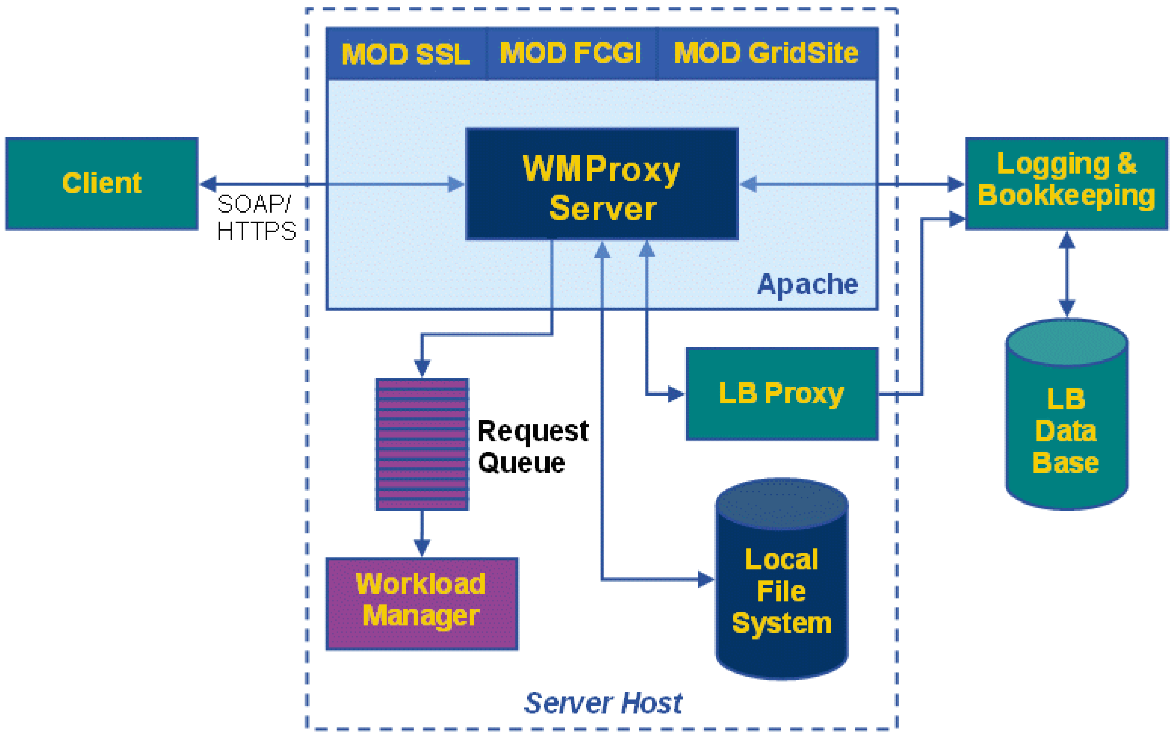
CGI can be seen as a new implementation of CGI, designed to overcome CGI's performance problems. The major implementation differences are:

- Fast CGI processes are persistent: Fast CGI applications are able to serve multiple requests
- Application instances are spawned/killed dynamically according to demand
- Fast CGI protocol multiplexes the environment information, standard input, output, and error over a single full-duplex connection.

WMProxy in its default configuration is run as a dynamic Fast CGI application in order to take full advantage of its capacity to accommodate request loads. However, it can be run in any of the available Fast CGI modalities. WMProxy is implemented as a single-threaded application thus ensuring more reliability and robustness than multi-threaded applications; thanks to the Fast CGI process manager the web server maintains a pool of processes that allow improving performances as generally done through multi-threaded application.

- **Grid Site** provides a module extending the Apache webserver for use within Grid frameworks by adding support for Grid security credentials such as **Grid Security Infrastructure (GSI)** and **Virtual Organization Membership Service (VOMS)**, and file transfer over HTTPS. It also provides a library for handling **Grid Access Control Lists (GACL)**.

The Web Service hosting framework provided by Apache, Grid Site and gSOAP has allowed the development of the WMProxy service using the C++ language. The choice of the implementation language has been driven mostly by the need to re-use and integrate existing production-quality components and libraries. A further boost in this direction has been given by preliminary tests that have shown better performance for the couple C++/gSOAP in comparison with the most commonly used framework for the development of Web Services (e.g., Axis/Java). Last but not least the gSOAP interoperability with a variety of other SOAP implementations and toolkits that permitted the development of a service accessible from clients written in different programming language and running on any computer platform.



The integration of the WMProxy within the WMS is shown in Figure 1. WMProxy provides a core module performing validation, conversion, environment preparation and information logging for each incoming request, before delivering it to the **Workload Manager (WM)**. WM is the core component of the WMS taking the appropriate actions to satisfy incoming requests. Communication between the WMProxy and the WM occurs through a thread-safe, file-system based queue. The LBProxy, which is used as a state storage of active jobs, is the only external service with which the WMProxy interacts. The LBProxy service provides an optimized access to the **Logging and Bookkeeping Service (LB)**. Other relevant information about processed jobs/requests are stored in the local file system in a reserved area managed by the WMS.

1.4 Security, AuthN & AuthZ

Message exchange between client and server is performed with SOAP/HTTPS.

Authentication of the requesting user is done through the **X.509 proxy certificate** signed by a trusted **Certification Authority (CA)**, which guarantees that the exposed public key is really owned by the user. The authentication process is handled by the Apache HTTP server by means of the SSL and Grid Site modules. The authenticated request together with information about the checked credential (e.g. expiration time, VOMS extensions), are then passed to WMProxy within the CGI environment. WMProxy does not need to directly manipulate Grid credentials in this phase.

Authorization is implemented through the **Grid Access Control List (GACL)** library, which is provided by Grid Site for manipulating **Access Control List (ACL)** files. Authorization can be either **Fully Qualified Attribute Name (FQAN)** (coarse-grained) or **Distinguished Name (DN)** based (fine-grained) according to the type of proxy presented by the client.

There are two authorization steps that are performed for an incoming request within the WMProxy:

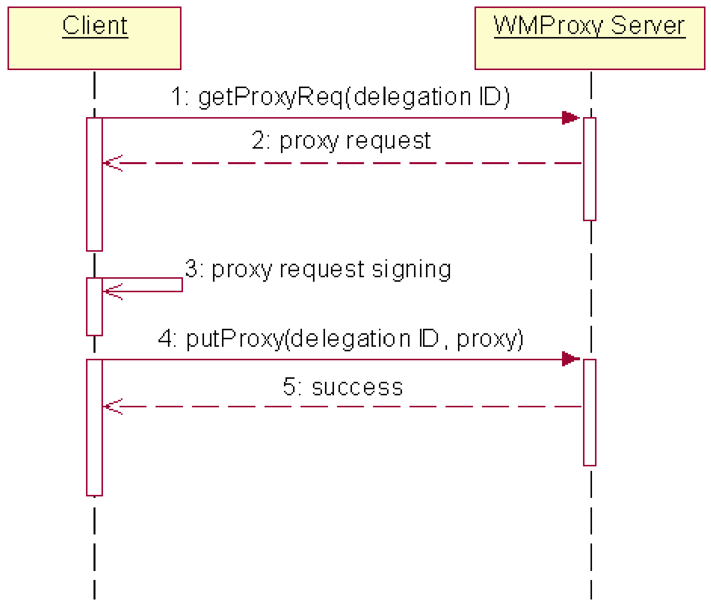
- Determine whether the requesting user is authorized to use the WMS services.
- Determine whether the requesting user is either the owner of the job or has permissions to manage it. (This step is only performed for operations directly related to a given job)

User Mapping: when a user performs a request to the WMProxy service, he is mapped to a local user, which is needed to perform all filesystem operations related to single jobs. This mapping is done through **LCMAPS** module.

1.5 Credentials Delegation

1.5.1 Delegation Process

The delegation is the process used to transfer rights and privileges to another party. Since the WMProxy and the WMS when providing some services need to interact with other services, operating on behalf of the user, a delegation process is needed to transfer client proxy credentials to the server host. The delegation service is provided through a port type whose description is imported into the WMProxy WSDL file from the gLite common delegation WSDL file. Delegated credentials are uniquely identified by the association of the delegation identifier, provided by user, and the user s DN within the credentials. Multiple delegations of the same proxy credential are allowed with different delegation identifiers; however, it is recommended to do it once at the beginning of the working session and reuse the same delegation identifier, as delegation process is generally time-consuming. The WMProxy holds a cache of the delegated proxies, which is purged periodically from the expired credentials; upon a submission request the service performs a mapping between the incoming job and a proxy in its cache according to the requesting user DN and the specified delegation identifier. From that point on, each operation performed for that job is done using the credential associated to it in this way.



1.5.2 Credentials Renewal

The User can store a long-lived certificate that can be used by the WMS in order to renew the lifetime of a standard user certificate proxy (usually valid only for 12 hours). Long-running jobs may run into this limit and fail due to expioration of user proxy. WMProxy can automatically request the registration for renewing the proxy certificate sent by the user. This is done through the attribute "MyProxyServer" inside the submitting JDL. When a Proxy Renewal Registration is requested for a certain Job, the WMProxy registers it to the proxy renewal. From that moment on, credentials for such jobs are granted by renewed certificate credentials. Actually all jobs that uses the same credentials links to only one renewed certificate.

1.6 Provided Functionality

The operations provided by WMProxy are listed below:

Version:

- **getVersion** - returns server internal version

Job Submission and Control:

- **jobListMatch** - finds resources which match job requirements
- **jobSubmit** - one shot job submission (job registration + job start)
- **jobRegister** - registers a job for submission
- **jobStart** - starts a previously registered job
- **jobCancel** - cancels a job from WMS
- **jobPurge** - clean-up of job local disk space
- **getOutputFileList** returns the URIs of job output files

Sandbox and local disk space:

- **getSandboxDestURI** - returns the input sandbox destination URI location for a job
- **getSandboxBulkDestURI** - returns the input sandbox destination URIs location for a group of jobs
- **getMaxInputSandboxSize** - returns the max size available for input sandbox files
- **getFreeQuota** gets the user available local disk space
- **getTotalQuota** - gets the user total local disk assigned space

JDL:

- **getJDL** - returns the JDL of a submitted job (original or registered to LB after pre-processing)

Proxy:

- **getProxyReq** - gets a request for a Proxy to delegate
- **putProxy** - put the delegated Proxy into the server machine
- **getDelegatedProxyInfo** - gets information of user delegated proxy
- **getJobProxyInfo** - gets information of the delegated proxy used to register/submit a job

Job's File Perusal:

- **enableFilePerusal** - enable Job's File Perusal functionality setting the file name list
- **getPerusalFiles** - get perusal files previously inserted in the list for perusal functionality (allows inspection of job s files while the job is running)

ACL:

- **addACLItems** adds jobs GACL file entries
- **getACLItems** gets jobs GACL file entries
- **removeACLItem** removes jobs GACL file entry

Templates:

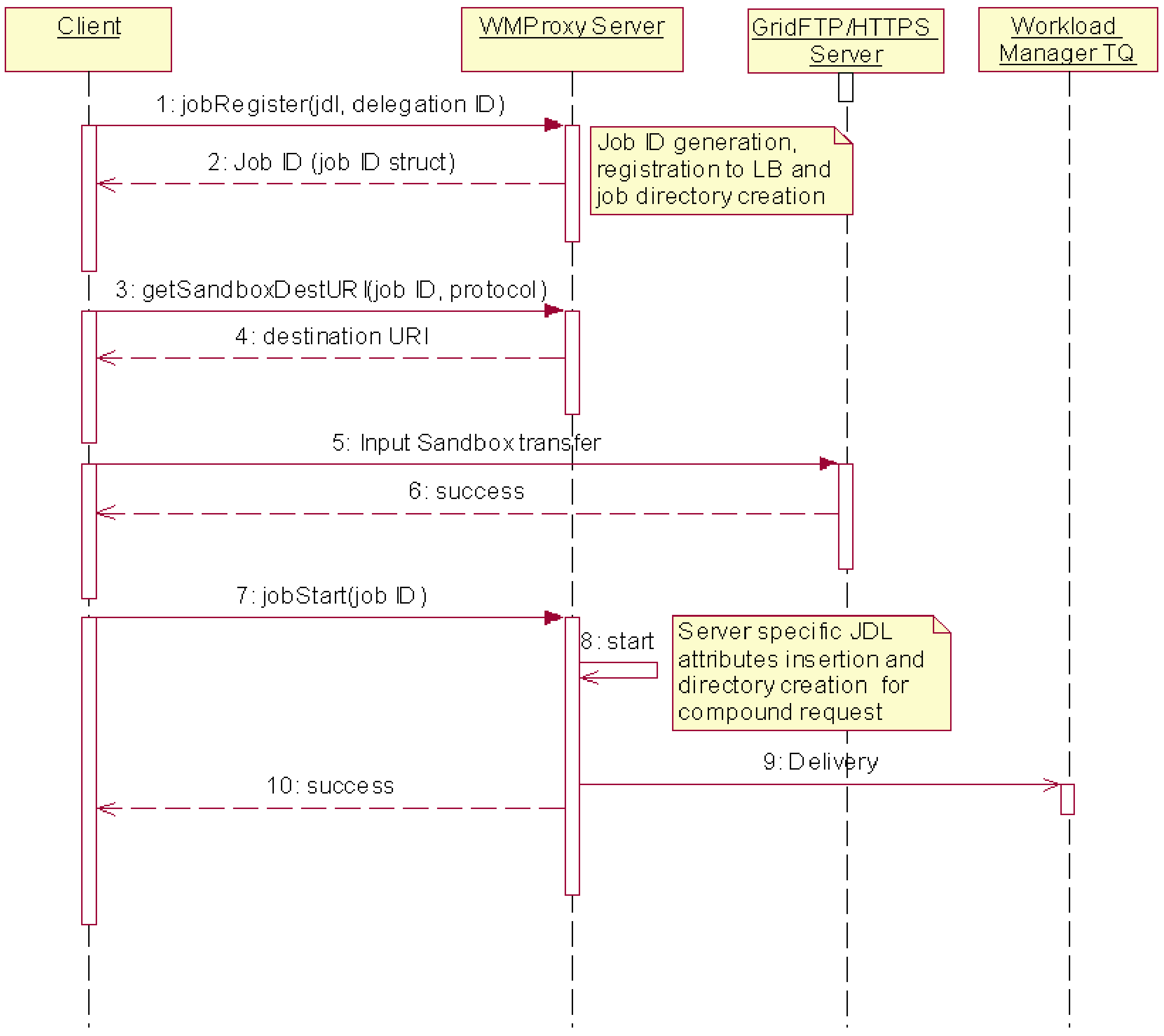
- **getJobTemplate** - gets a template for a simple job
- **getDAGTemplate** - gets a template for a DAG
- **getCollectionTemplate** - gets a template for a collection
- **getIntParametricJobTemplate** - gets a template for a parametric job with integer parameters
- **getStringParametricJobTemplate** - gets a template for a parametric job with string parameters

Transfer protocols:

- **getTransferProtocols** - get the server currently available file transfer protocols

1.7 Job Submission

In this section we describe the sequence of WMProxy operations and the corresponding actions that need to be performed to accomplish job submission to the WMS, i.e. the delivery of the user s job request in the WM request queue. Job submission requires both, a description of the job to be executed, and a description of the needed resources. These descriptions are provided with a high-level language called JDL. The actual submission is done by calling in sequence the two service operations jobRegister and jobStart When a jobRegister request arrives to the WMProxy, if the client has the rights to proceed, a job identifier is generated and a set of specific attributes needed by the WMS for handling the request appropriately are inserted in the job description. The requesting user is then mapped to a local user by means of LCMAP so that the job local directories and files can be created with appropriate ownership and permissions. When all the aforementioned steps have been successfully completed, the job with the generated job identifier and the enriched JDL description is registered to the LB and from that point on is uniquely identified and can be followed-up throughout the whole system with its identifier. Afterwards, the job is registered for proxy renewal (if requested) and the job handle is returned to the user. In case the JDL description provided by the user represents a compound request (see next section), i.e., a set of jobs, a tree of identifiers is returned. This tree contains the job identifier of the single jobs composing the group, and a job identifier to identify and manage the entire group. The completion of the WMProxy job registration phase is the sign that the job has been taken into account within the Grid environment and is ready to be actually submitted to Grid resources. This further step is triggered by the invocation of the jobStart operation. During the start operation the provided job identifier is used to check if the corresponding job has the pre-requisites for being started, i.e., it has been previously registered by the same WMProxy service and it has not already been started. Only if both conditions are met or previous job start attempts have failed for some reason the job processing is carried on by further manipulating the job description, decompressing, extracting, locating sandbox files in the appropriate areas and lastly writing the request into the WM requests queue. Compound jobs start requires additional steps to be performed before passing the request to WM. These are the registration to LB of all sub-jobs composing the request and the creation of the job reserved area on the file system for all of them. All auxiliary actions needed for the job to run successfully on the remote resource can be performed in the phase between job registration and job start. Relevant examples could be the preparation of the input files needed by the job at run time including both the job input sandbox files and the data stored on **Storage Elements (SE)** and registered onto **Grid Catalogs**. In particular the job input sandbox files can be either located on accessible external file server or have to "travel" together with the job from the client machine to the resource where the job is going to run. In the latter case the needed files have to be uploaded by the user to the WMS node so that they can be then managed by the WMS and made available to the job. WMProxy provides the URI of the files destination location through specific operations. Supported file transfer protocols are gsiFTP and HTTPS. The split of the job submission process into the two well distinct phases, job registration and job start, realizes a sort of two-phase commit: with the former operation the job enters the system remaining registered for a configurable period of time; with the latter one, the job can utilize the system. This gives the user full control over single submission phases; in case of failure single operations can be performed again separately. Moreover it allows performing preparatory activities when the job has already entered the system and moving specific time-consuming processing after the operation response has been provided to the client.


Client
WMProxy Server
GridFTP/HTTPS Server
Workload Manager TQ
1: jobRegister(jdl, delegation ID)
2: Job ID (job ID struct)
Job ID generation, registration to LB and job directory creation
3: getSandboxDestURI(job ID, protocol)
4: destination URI
5: Input Sandbox transfer
6: success
7: jobStart(job ID)
8: start
Server specific JDL attributes insertion and directory creation for compound request
9: Delivery
10: success

1.8 Service Interface

The WMProxy service exposes operation through a WSDL interface:

http://web.infn.it/gLiteWMS/index.php/techdoc/howtosandguides

The most important purpose of SOA is to achieve loose coupling among interacting components allowing implementation of autonomous services. The loose coupling is achieved through a small set of simple and ubiquitous interfaces, and a set of descriptive messages. In the interfaces only the generic semantics are described: the necessary semantic needed to anticipate possible server behaviour. These interfaces are theoretically available to the global Internet community. The messages represent the way of communication between server and client, and must be descriptive and non-instructive: the server is the only one responsible for the specific design and implementation of the provided service. The messages must be also extensible to allow changes in the service, if needed. To adhere to the SOA model during design and implementation of the WMProxy, the previous guidelines were adopted trying at the same time to provide good service performances to fulfil the user needs for Quality of Service (QoS). Since the WMProxy is a SOAP Web service, the interface is described through the WSDL. A WSDL file represents the available interface needed by the client to access the service. In this file all the operations (services) provided by the WMProxy are described with their argument list, the return value (or structure) and the fault messages. The WSDL file was written following the WS-I Basic Profile. This profile defines a set of Web service specifications that promote interoperability. WS-I Basic Profile addresses the most common problems that could generally affect interoperability, although conformance to it does not completely guarantee the interoperability of a particular service. The profile defines a basic level of conformance based on artifacts. There are three kinds of artifacts, these are:

- **MESSAGE**
- **DESCRIPTION**
- **REGDATA**

WMProxy takes into account the **DESCRIPTION** artifact, i.e., the descriptions of types, messages and interfaces with protocols, bindings and access points are included in the WSDL file. Compliance of **MESSAGE** artifacts is assured by the used SOAP container (gSOAP), while the **REGDATA** principally refers to service registration and discovery with services like **Universal Description, Discovery and Integration (UDDI)**. The latter aspect has not yet been considered exhaustively as it was not a requirement in the EGEE project where a proprietary service discovery mechanism is used. Besides types, messages and interfaces a WSDL file also describes their protocol, data format bindings and the network access points. An instance of a Web service (or a port, represented inside the WSDL file with the wsdl:port tag) is considered conformant if it produces conformant artifacts, and is capable of consuming conformant artifacts, as appropriate. The profile considers all the instances defined for the service (the ports), and verifies the conformance for any of them. Types of Web services (wsdl:binding and wsdl:portType tags) are considered conformant if they result in conformant instances. These are basic rules completed by profile normative statements. In general, to implement a WS-I compliant service, the profile requires a WSDL file, which is composed of specific parts and both describes the service and follows a mandatory structure. The WMProxy WSDL file defines two different ports to provide WMProxy-specific and delegation services (described through an imported WSDL file). This delegation service is the standard service used throughout all gLite, and it is being modified to be aligned with the **GT4** delegation service. WSDL type definition of operation arguments is provided to guarantee the correct behavior of all service operations, when accessed from varied language implementations of the requesting client (interoperability). Programming languages have different ways to treat with operation arguments, and different emitters create specific kinds of stub and skeleton (proxy). In the WSDL file, fault messages are also described. These messages are sent back to the client in case of malformed client requests, or in case of service failure. To handle them with a simple, structured approach, an ad-hoc XML Schema type has been designed: the BaseFaultType. The specific fault complex types are developed to extend the BaseFaultType type, and any fault is handled as a BaseFaultType fault at higher level. Work is on-going to make the BaseFaultType definition fully compliant with the WS-BaseFaults specification.

-- FabioCapannini - 2011-11-04

This topic: WMS > WMProxyDocumentation
Topic revision: r5 - 2011-11-16 - FabioCapannini