Data and Metadata Management

Project Number:	FP6-2005-IST-026996
Deliverable id:	D 9.3
Deliverable name:	Data and Metadata Management
Date:	May 19, 2008

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 9.3
Document name:	Data and Metadata Management
Document type (PU, INT, RE)	INT
Version:	0.8
Date:	19/05/2008
Editors: Organisation: Address:	Stelios Sfakianakis FORTH-ICS Foundation for Research and Technology-Hellas (FORTH) Institute of Computer Science N. Plastira 100, Vassilika Vouton, GR-700 13 Heraklion, Crete, Greece

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

ABSTRACT:

This document aims to provide information on the data and metadata management capabilities of ACGT. In particular it presents a review of the different kinds of data handled in ACGT, the use cases, and the infrastructure components for managing their lifecycle, annotation, and discovery. The Grid Data Management Services infrastructure is described and its use for handling user managed data objects is presented. Additionally the metadata repository and its services are used for storing, indexing, and maintaining the annotations of non data entities such as services and workflows.

KEYWORD LIST: data; metadata; metadata registry; semantic interoperability; integration

MODIFICATION CONTROL			
Version	Date	Status	Author
0.1	15/03/2008	Draft	Stelios Sfakianakis
0.5	12/05/2008	Draft	Stelios Sfakianakis
0.7	18/05/2008	Draft	Stelios Sfakianakis
0.8	19/05/2008	Final	Stelios Sfakianakis

List of Contributors

- Stelios Sfakianakis, FORTH-ICS
- Marcin Wolski, PSNC
- Pawel Spychala, PSNC
- Maciej Labedzki, PSNC
- Johan Karlsson, UMA

Contents

Table of Figures

Acronyms

API	Application Programming Interface
BPEL	Business Process Executable Language
CA	Certification Authority
DMS	Data Management Service
GSI	Grid Security Infrastructure
GT4	Globus Toolkit 4
GUI	Graphical User Interface
HTML	HyperText Markup Language
RDF	Resource Description Framework
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
VO	Virtual Organization
WS	Web Services
WSDL	Web Service Description Language
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

Executive Summary

In modern system architectures data is the most valuable property that should taken care of and handled. The World Wide Web is an example of such huge data oriented architecture and ACGT aims to provide similar data handling capabilities but to a lesser degree and with more stringent control and discipline. This document aims to provide information on these data and metadata management capabilities of ACGT. In particular it presents a review of the different kinds of data handled in ACGT, the use cases, and the infrastructure components for managing their lifecycle, annotation, and discovery. The Grid Data Management Services infrastructure is described and its use for handling user managed data objects is presented. Additionally the metadata repository and its services are used for storing, indexing, and maintaining the annotations of non data entities such as services and workflows. The general framework and requirements are laid down and we expect that during the course of the project and as more experience is gained through the system's use this document will become more complete and detailed.

1. Introduction

Data is the ACGT's most important asset: it gives value to the platform and empowers the users in exploiting the infrastructure. Data are included in every user visible interaction within the ACGT platform:

- Data in the form of CRFs are designed and later on persisted by the ObTima system
- Data are modeled and formally reasoned about in the form of the ACGT Master Ontology for Cancer
- Data are queried, searched for, and retrieved by the Semantic Mediator and the underlying Data Wrappers and anonymized databases infrastructure
- Data are analyzed, evaluated, mined, interpreted, etc. by the Knowledge Discovery tools and services
- Data are produced as the results of the enactments of user specified Scientific Workflows and as by-products of the users interactive sessions with the overall ACGT system
- Data are exposed in the form of metadata to give both machine interpretable and user perceived descriptions of the kind, the functionality, the quality, and other aspects of the ACGT software actors (databases, services, tools)

The management and curation of the data is therefore vital for the platform's well being and in essence it is the data handling and processing capabilities of ACGT that are the driving forces for the project's success.

Since the data is of great importance for ACGT it is required that a lot of effort should be devoted to the design and the implementation of the infrastructure that is needed for the management, handling, and preservation of data. In an effort to give a high level overview of the data and metadata management use cases we can identify the following responsibilities of the ACGT data management layer:

- **Identification**. Identity is a vital property of any managed entity. By the means of identification the system can catalogue and uniquely name distinct objects. It further allows the addressing of these data objects and their efficient composition in even more complex objects (e.g. graphs) by incorporating these unique identifiers instead of the contents of the objects themselves. Furthermore a data store using a data identification scheme can be efficiently modeled, organized, and searched.
- **Classification**. The process of classification allows the assets to be grouped in meaningful ways. Classification can be based on data attributes such as the domain of use, the producer or the consumer, the intended purpose, etc. Data therefore can be classified in a multitude of ways and the user or the system could exploit the multiple classifications to support browsing and searching capabilities.
- **Valuation** Not all data are equal and not all of them are equally important. For example specific precautions should be taken for handling patient sensitive information whereas performance logs of a software component do not have the same confidentiality and risk analysis requirements. Data should be evaluated with respect to their specific requirements and classified and managed accordingly.
- **Depreciation**. In essence the value of the data does not remain constant. Some data's quality is affected by time: user interests are one such example where changes in the area that one works or is active can devalue the importance of previously expressed interests. A system should be agile and flexible on the way it values or devaluates its data assets as its environment changes.

- **Maintenance**. As a result of the changes in the environment and the participating actors (e.g. users, software entities, etc.) data need to maintained and preserved under the new conditions. Maintenance ensures or extends the useful life of an entity and needs to be constantly performed so that data is kept consistent and correct.
- **Replacement or Destruction**. It may be the case that some data objects ultimately cease to exist or become irrelevant for their initial purpose. At this point a decision must be made to replace or destroy them, depending upon whether or not they are still needed. Replacement with a newer version of the data may be adequate if such an updated version exists. In either case the system should cope with references to the deprecated data and it may be the case that a newer version should coexist with the old one, for example for keeping the history.

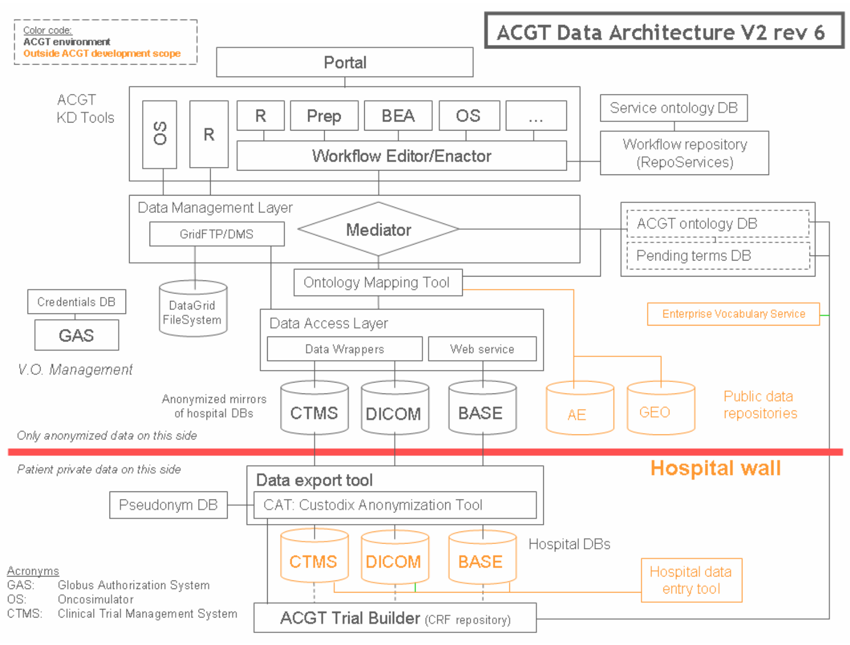

Figure 1 The Data Architecture of ACGT

A carefully designed system should deal with all these dimensions of data handling. In this document we present the technologies employed for the implementation of the various aspects of data management described above. Not all of these aspects are relevant to every use case scenario that involves data but nevertheless they give a general framework for the discussion.

Data Management Overview

The ACGT platform is a cross pollination of Grid infrastructure, Service Oriented middleware, and Semantic Web technologies. The aim is to get the most out of these methodologies in a complementary way so that the final system offers efficient storage and management of data, intelligent search and discovery of data and services, and secure processing of data to extract information and knowledge. In this setting data are ubiquitous and of paramount importance and therefore the platform needs to take care of their whole lifecycle that described in Introduction: from the acquisition and generation of new data, the storage, the identification and annotation, the quality control, and finally the destruction or replacement of them.

In this section we give a high level overview of the different aspects of data and metadata management.

Data Origination and Stewardship

The ACGT system deals with data that possibly originate from different data sources. Some of them are external to the ACGT while others are parts of the architecture. In the case of the external data sources there are cases where the data are managed outside of the ACGT area (public databases) and therefore the platform does not deal with the issues involved for their data management. Therefore we are especially interested in data that the ACGT system is responsible for and an important question to begin with is the following: *How's the data produced in the first place?* In trying to answer this question we can distinguish two cases:

1. The data are produced outside of ACGT and then imported into the system, or
2. The data are produced within the ACGT platform.

This distinction is important because the requirements of each case are different. In the first case the platform should provide the needed framework [D11.2] for the proper anonymization and protection of sensitive information so that no security related issues are introduced. In this case it is also more important that the input data are evaluated in terms of their quality and the user is facilitated in order to provide meaningful metadata descriptions. In the latter case the ACGT system as a whole is more in control of what is produced and hence the data evaluation and annotation tasks are easier.

Another distinction we can make is whether the user is “responsible” of the data. Responsibility here refers to *stewardship* i.e. the supervision, maintenance, and the lifetime management of the data. As one would expect integration of data objects that the ACGT platform is responsible for is more stringent and better controlled, for example the system can register their lineage and provenance information in detail. Nevertheless these and the previous classifications can be highly intermixed and in particular we can identify the following use cases for data generation:

1. Patient specific data are stored in databases that are introduced in the ACGT architecture and become part of its data layer. Actually what is entered in the ACGT are “copies” of these databases that mirror the original contents with the only difference being that the data are anonymized (see Figure 1). These mirror databases are accessed through the corresponding Data Wrapper interfaces and logically joined and queried together by the Semantic Mediator using the high level concepts and relationships of the Master Ontology. The ACGT system in general and the Virtual Organization authority in which they participate in particular are considered to be the curators of these databases.
2. The data are uploaded by the users and kept private on their ACGT storage area or even made public if their creators decide to do so. This is a specialized case of the previous one but has some distinct requirements: the user should be provided with

the anonymization and other tools to perform data cleansing and the anonymization herself and she is on charge of the management of the uploaded data. These data are by default private and are not accessible by the mediator in part because of the privacy issue but mostly because the mediator does not integrate single data files.

3. The data are produced by the analytical and knowledge discovery ACGT tools and services when existing data are processed in user submitted workflows. In this case data are produced inside ACGT and they are again in the hands of the users that created them by initiating the data processing task.
4. The data are produced by monitoring the users' actions and the utilization of the system in general. This kind of data can be used for building user profiles or making intelligent conclusions about the way the platform is used or even for discovering dysfunctional behavior and if the system in underperforming.

Data Storage and Maintenance

There are different storage means and services for the different kind of data that described above. The anonymized ACGT databases are the primary source of data that were introduced in the ACGT system. These databases are mirrors of their corresponding external ones are described above and they are usually managed in the context of a Virtual Organization (VO), which means that access to them is granted by the VO administrators as described in Deliverable 11.2 [D11.2]. For these data sources the maintenance, depreciation, valuation, and replacement/destruction tasks are performed by the VO managers according to the requirements of the corresponding Clinical Trials. From the engineering point of view the maintenance of these databases includes getting up-to-date copies of the external (primary) databases and going the same steps of anonymization and their publication in the ACGT platform. The identity preservation is important so that new replicas of the data replace the old ones and existing correlations (e.g. through the patients' pseudonym) remain valid after the introduction of the new versions. The implementation details vary on a per-database case and can be dealt with in a VO specific way.

Each of these databases exposes an OGSA-DAI programmatic interface through the Data Access Wrappers. The notion of identity and classification is supported in the global level by the Semantic Mediator and the Master Ontology on Cancer. In particular these technologies use URIs for the identification of specific data items while semantic concepts drawn from the Master Ontology are attached to them so that categorization and retrieval are supported in a manner that abstracts away the particularities of the underlying databases.

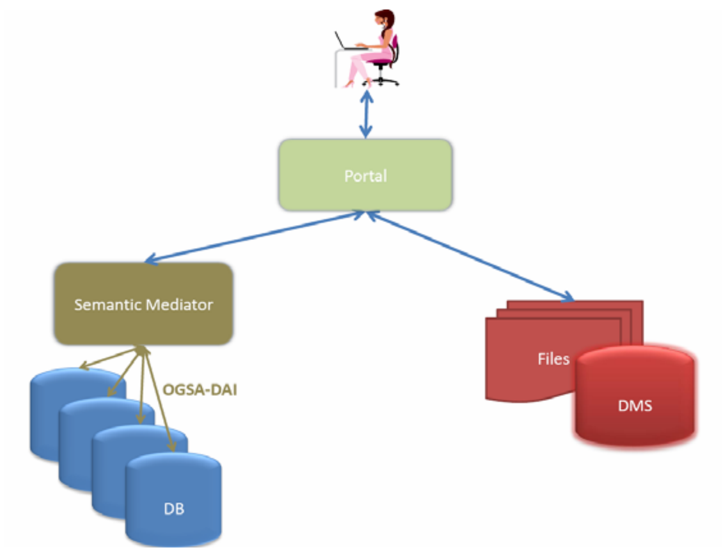

Figure 2 Accessing Data in ACGT

On the other hand the data produced in the context of a workflow or the data uploaded by the user are handled differently. These are not integrated by the semantic mediator and are in the control of the user who created or uploaded them. Ideally the Semantic Mediator could be enhanced to integrate also this data but there are various issues that prevent us to do so:

- The user could keep these data for herself in a private ACGT store and not share with anyone else
- Even if the user makes some of this data public (or shared in the context of a VO) the architecture of the mediation process would put a lot of burden to the user because she should provide a mapping of the data to the Master Ontology concepts in the format used by the Mediator
- The Semantic mediator is designed and implemented under the assumption that the underlying data sources are databases that exhibit a more or less relational schema[1] while the data provided by the user are in the form of files (which can be of any internal format).

For the storage of these user provided data files the Grid Data Management Services (DMS) are employed. Basically each ACGT user has her own storage area on the Grid where she can create, retrieve, update, and delete (CRUD) data. For this use case the user is in charge of maintaining and managing her data sets and the ACGT services aim to support the users on this context. On the other hand the ACGT system and in particular the Grid data services are responsible for the provision of a generic file system that supports “durability” (i.e. the data are persisted) and file identification through its Grid logical and physical file names. More details about the Grid Data Management layer are provided in the section 2.

So in conclusion the major storage means for data are the ACGT databases and the Grid data management facilities: Data which are provided by external databases in the context of a VO are stored in the ACGT mirror databases that are accessible by the mediator while data created by the users within ACGT or uploaded by them are stored in the Grid “file system”. The users have access to both kinds of data sources through the Portal as shown in Figure 2.

Metadata

Metadata are a special type of data that comprises the information that is pertinent to an entity without being the entity’s “content” per se. Metadata can vary from simple *descriptive* annotations, such as the creator of a file or its last modification time, that provide more context and explanation about the entity described, to more high level information used for the *interpretation* of the data, for example the format used, the purpose of the data, or in abstract terms its *meaning*. In ACGT we foresee the use of metadata of various artifacts:

- Data sets stored as files and databases
- Services and tools invoked over the network
- Workflows, i.e. user scenarios that consist of multiple existing data sources and data processing elements working together
- Results of workflows and services invocations

Some of these metadata are provided by the users, such as descriptions for their workflows, while the majority of the metadata are produced by the system itself or entered by the administrators and the system developers, for example the services implementers. Nevertheless

[1] Or a schema that could be exposed as relational such as the DICOM information model.

the metadata are again one kind of data that needs to be stored, discovered, retrieved, and maintained.

On the other hand metadata are special in the sense that they usually exist in relation to some other data. It is not reasonable to have metadata describing or annotating nonexistent data or to have metadata that are not linked to data. Such linking and association of data and metadata is essential for supporting use cases where the following questions are raised:

- “What do these data mean"? i.e. given a data set I want to locate its metadata
- “I am looking for information items with these specific characteristics”, i.e. given some metadata I want to locate data sets that have this "meaning"

The way that data and metadata are associated and how this association is maintained is critical for supporting the use cases above and basically there are the following possibilities:

- **Metadata embedded in Data**. In this case the metadata are embedded in data and are directly accessible through the data. An example of this approach is the Semantic Annotations for WSDL proposal[SAWSDL] where the semantic annotation of a web service interface is hosted inside the WSDL description of the service. Also microformats[2] and RDFa [RDFA] are relevant technologies that support embedding metadata in XHTML documents, which can be extracted later on, for example as RDF descriptions using GRDDL [GRDDL]. In this approach it is evident that locating the metadata of some data is very easy because they are readily available in the data payload. The inverse question though, i.e. searching for data based on metadata, requires additional machinery (either some central registry or a semantic crawler or a combination of both).
- **Metadata served with the data**. A variant of the previous case is having the metadata wrapping the data itself in a kind of semantic (or other) envelope. Typical examples of this case are the headers of the HTTP messages (e.g. Content-Type) or the head element of an HTML page where we can have metadata such as the ones provided by Dublin Core. This approach requires either some extensible data formats that provide such generic envelopes (e.g. SOAP) or network and communication protocols where metadata can be prefixed (or appended) to the data and transferred in the same interaction and through the same medium.
- **Metadata associated with data through external means.** In this approach the metadata are provided outside of the data and there is no linking information in the data itself. An example of this approach is a UDDI registry that provides information for a Web Service such as a business category for the service (in some classification scheme) and contact information. This approach is actually that of a Metadata Registry (such as the one described by ISO 11179) that is the single authority for maintaining metadata descriptions and associations and also answering queries based on metadata.

The connection between data and the corresponding metadata is important and therefore the first two approaches are usually considered as the best. Especially embedding the metadata into the data itself provides an always up-to-date version of metadata and the synchronization between the two degenerates to a “no-op”. On the other hand efficient search for data based on metadata usually requires indexing that cannot be performed “auto-magically” inside the data items themselves. Hence we have followed a hybrid approach where the majority of metadata management is supported by specialized registries. So for the administration and manipulation of the metadata we have defined the following components in the ACGT architecture:

[2] http://microformats.org/

- Metadata Repository and its access services
- Grid Data Management services
- Workflow Repository with enactment and provenance information

The Metadata Repository is currently used for storing metadata about services, workflows, R scripts. The Grid DMS also features metadata support for the files managed. So currently there's a clear separation about the storage of metadata:

- For the file related metadata the DMS is to be used. This means that data uploaded by the users or produced by the enactments of their workflows are annotated with metadata which are persisted by the DMS
- For the "active" content, i.e. services and workflows, the Metadata Repository is used both as a registry and a "yellow page" directory. The service related metadata therefore are stored in the Metadata Repository.
- Provenance information i.e. what was produced by whom and from what input is stored in the Workflow repository.

This separation of the service, data, and provenance related metadata was decided because the data which the users are in control for are stored in the DMS anyways and therefore it's better for consistency reasons to have a single point to store both the data and their metadata. Additionally the file related metadata are in flux at the current stage of development and the metadata repository was decided to be kept as stable as possible in its development roadmap due to its use as a central registry of the available services and tools. Finally the workflow repository is somewhat in the middle of the two other repositories in the sense that it relates the service and workflow static information, which persists in the metadata repository, with their results, which are stored in the Grid data management facilities, through the means of the enactment information, which is dynamic in nature, that is maintained in the workflow repository. Linking and correlation between these repositories is achieved by the means of the unique identifiers supported by each repository. Nonetheless, a unified repository for metadata could be implemented in the future.

The DMS provides an example of metadata that are handled and served in close connection with the data and it's described in Section 2. Finally the Metadata Repository is briefly described in Section 3.

2. Grid Data Management (DMS)

The Data Management System is a distributed system featuring data access and delivery. It is based on SOA model, which is an architectural style of building software applications that promotes loose coupling between components so that you can reuse them. According to those definitions, each DMS module can be treated as a separate service using XML messages and passing them in a form of SOAP envelopes to external applications.

DMS is a kind of a virtual file system keeping the data organized in a tree structure. The main units of this structure are meta-directories, which make it possible to catalog other objects and meta-files. Meta-files represent a logical view of computational data regardless of their physical storage's location. The computational resources managed by DMS can be described by metadata, which create an abstract, semantic and explorable layer of resources. The flexible access to data requires DMS to support various transmission protocols – DMS enables to store and share data files through the FTP, GridFTP, and FTPS protocol.

DMS does not duplicate the databases features – the basic concepts which lied down at the its creation were the development and implementation of the environment allowing flexible access to data stored inside the system, accessible by the most popular protocols and opened to standards defined by grid organizations.

The Data Management System contains three logically distinguished modules: Data Broker (***DB***), Metadata Repository (***META***) and Data Container (***STOR***), which together create the basic layer of the data management environment, the so-called DMS core. Each system module is treated as a separate network service communicating with the external environment with the help of the SOAP protocol. DMS provides its services in a form of Web Services API to the front-end applications. It is important to mark the DMS position in an open grid environment – DMS is a middleware system, belonging to the collective layer as well as the resource layer (Data Container), according to the grid services view.

Basic functional assumptions

This chapter presents basic concepts related to the file managing in the DMS environment. It brings closer the main functional assumptions for files and directories and presents a few guidelines for storing and retrieving data. Moreover, it explains in detail how file state changes according to different type of operations executed on them. A separate section is devoted to a metadata schema, which makes it possible to describe the semantics of DMS resources.

Fundamental features

There are a few fundamental features, which are common for all groups of services belonging to the DMS interface. These are:

- Operations in DMS are executed in the user context. That means that if e.g. the Data Broker receives a request to return a list of directory elements, it is equivalent with the fact that the user with the given ID sent that request.
- All objects in the DMS are identified by their unique identification numbers (ID).
- Data Broker checks the user rights before any operation on a given resource is executed.

- During typical workload some exceptional situation may occur. All of these have the unique identifier describing the error class, which they belong to.

Data management

The following section introduces the basic functional assumptions related with data services, that is storage and sharing of data. These are:

- The node name - the node name has to start with an ASCII character with number greater than 32. ASCII character number 47, that is '/' can't appear in the name, because it is directory separator. The length of a name can't be greater than 128 characters.
- The root directory - there exists a main directory (without a parent), which is the root of a metadirectory tree. It is called `root`.
- Names uniqueness - there can't be two elements with the same name in one metadirectory.
- URL standard - the services connected with storage and access to the files are based on URLs, which specifies the place where a file is stored unambiguously. The structure of such an URL is described in RFC1738 specification and has the following form:

```
<protocol>://<user>@<password>:<host>:<port>/<full path>

http://storage01.dms.progress.psnc.pl:3043/./data/6d7is3ao6e7d.888
```

- Data Container transparency - DMS chooses data container independently, basing on the up-to-date knowledge about the active containers, and stores the file in it.
- The process of uploading data file in the DMS contains the following three steps:
  - Creation of the logical instance of the data file in the Metadata Repository;
  - Reservation of the disk space in the Data Container, parameterized by the file size, the reservation lifetime and optionally the preferred protocol;
  - Data file transfer to the Data Container using its external transport server (e.g. FTP server).
- The data file downloading process contains two phases:
  - Request of an access to the data file from Data Container resources parameterized with the reservation lifetime and optionally protocol type;
  - Data transfer to the client machine from the Data Container through the appropriate transfer protocol (e.g. HTTP)
- Transfer protocols - files stored on Data Container are accessible using all the protocols supported by this container.
- File states - there are four possible states of a file:

  - **BARE** – a state of the file after adding it to the DMS. It indicates that the file doesn't exist physically on any server, nor it is being transferred to one;
  - **IN_TRANSFER** – a file is being transferred to the server and doesn't exist on any server yet;
  - **PERMANENT** – a file is actually on at least one server. URL to this file can be obtained and thus it can be downloaded. Note that the file in this state can currently undergo replication process, which preclude the deletion of the file;
  - **INACCESSIBLE** – file is locked. Probably it is being used by another process.
- On-demand file replication - user can initialize file replication at any time. File replication increases safety of the stored data.

Metadata management

The following section introduces the basic functional assumptions related with metadata services, which are used primarily to describe the resource stored in DMS. These are:

- Metadata can be assigned to any DMS object (file files or directories).
- Predefined metadata set - the Dublin Core is a predefined metadata standard in the DMS. Moreover, defining own file description schemas is possible.
- Internationalization - the metadata structure is multi-languages oriented. Schemas and their attributes always have a description in a default language version, which is created along with the schema. Additional language versions can be added later on.
- The metadata schema structure contains the following fields:
  - *Name*: mandatory field with maximum length of 200 characters;
  - *Description*: optional field with maximum length of 1024 characters;
  - *Version*: optional field with maximum length of 10 characters;
  - *Authority*: optional field with maximum length of 100 characters;
- Each schema can contain 0 or more attributes. Structure of its attributes is following:
  - *Name*: mandatory field with maximum length of 150 characters;
  - *Description*: optional field with maximum length of 1024 characters;
- It is possible to assign schema to the DMS object (data file, directory) by setting the schema attributes values for this object. Each DMS object may be described with few schemas, but one schema may describe only one DMS object. The value of the schema attribute may be empty, but if it is set its maximal length is 200 characters.

File states

The present section describes how file state changes according to the operations executed on them. The possibilities include uploading, updating and replication of a file. The changes of the file states differ when the file is zero sized

Uploading a file

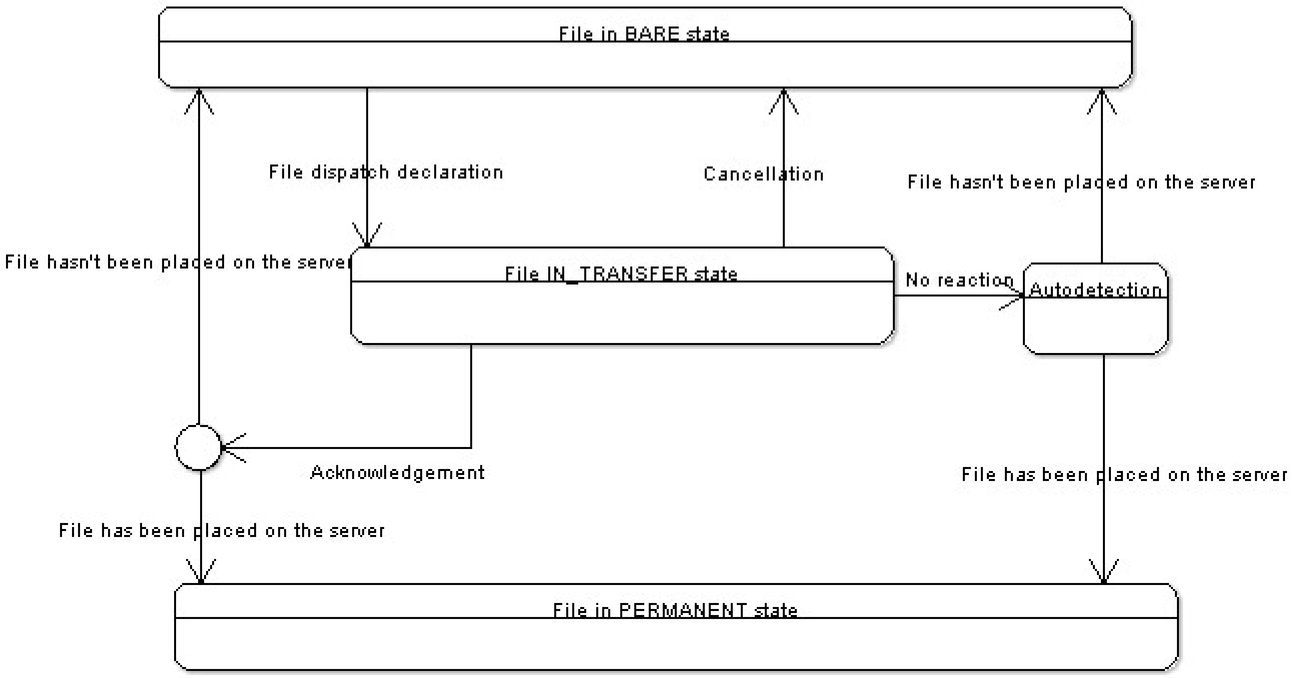
File in BARE state
File dispatch declaration
Cancellation
File hasn't been placed on the server
File hasn't been placed on the server
File IN_TRANSFER state
No reaction
Autodetection
Acknowledgement
File has been placed on the server
File has been placed on the server
File in PERMANENT state

Figure 3 File states during uploading operation (size <> 0)

Initially the file is in a BARE state. A request to create a physical instance of a file together with a lifetime of the reservation and estimated file size is delivered. The file goes into a IN_TRANSFER state. The following possibilities arise:

- Cancel of the file transfer – all the reservations are canceled, file returns to initial state – BARE;
- Confirmation of the transfer ending – Data Container checks, whether the file has been actually put on a server (i.e. at least one byte is written). In that case the file goes into PERMANENT state, in opposite case it returns to a BARE state;
- No reaction – Data Container automatically checks the transfer state after lifetime of the reservation. If there are no locks on the file at the moment (i.e. it's not being transferred at the moment) the operation is finished. Data Container checks, whether the file has been actually copied on a server (i.e. at least one byte is written). In that case the file goes into PERMANENT state, in opposite case returns to BARE state.

Uploading a zero-sized file

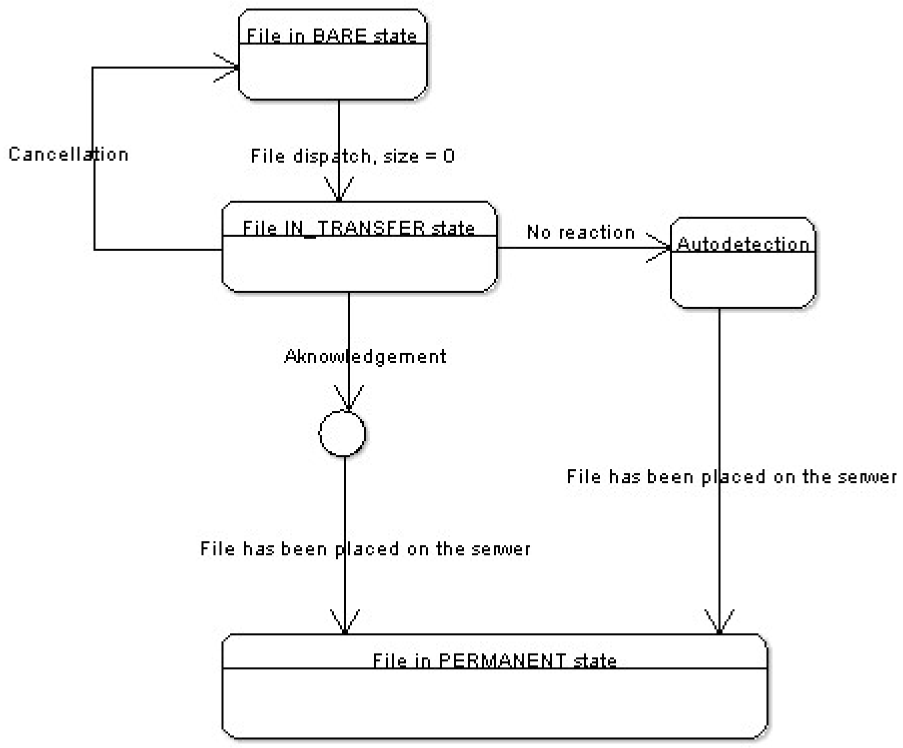

Figure 4 File states during uploading operation (size = 0)

Initially the file is in a BARE state. A request to create a physical instance of a file together with a lifetime of the reservation and estimated file size is delivered. The file goes into the IN_TRANSFER state. The following possibilities arise:

- Cancel of the file transfer – all the reservations are canceled, file returns to initial state – BARE;
- Confirmation of the transfer ending – the operation is finished and the file goes to the PERMANENT state no matter how many bytes of data were transferred;
- No reaction – Data Container automatically checks the transfer state after the lifetime of the reservation. If there are no locks on the file at the moment (i.e. it's not being transferred at the moment) the operation is finished. The file goes to the PERMANENT state no matter how many bytes of data where copied.

Updating a file

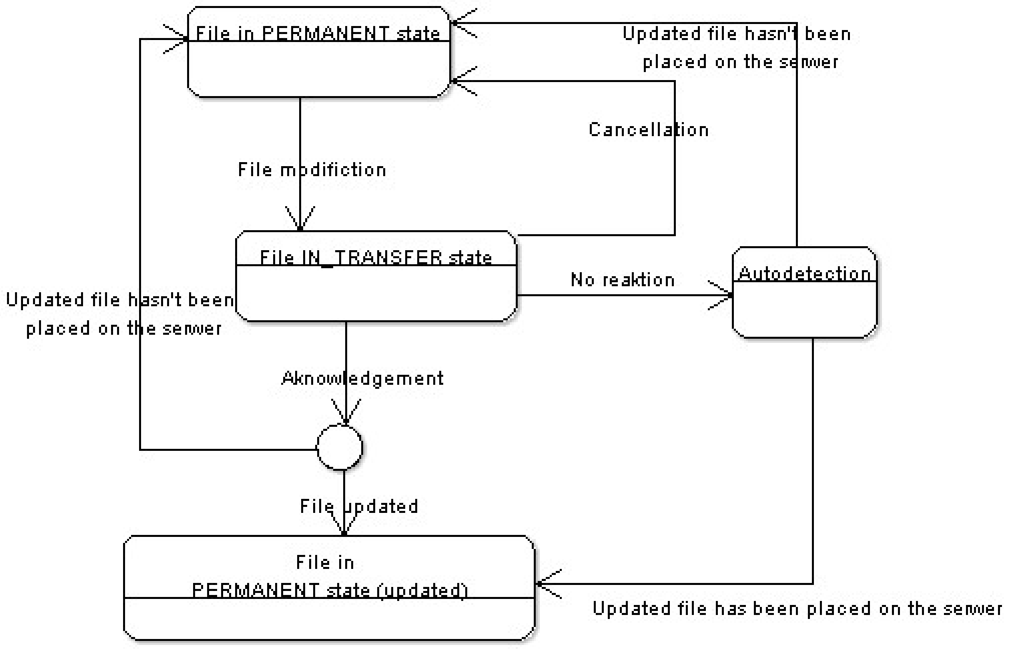
File in PERMANENT state
Updated file hasn't been placed on the server
Cancellation
File modifiction
File IN_TRANSFER state
No reaktion
Autodetection
Updated file hasn't been placed on the server
Aknowledgement
File updated
File in PERMANENT state (updated)
Updated file has been placed on the server

Figure 5 File states during updating operation

Initially the file is in a PERMANENT state. A request to create a physical instance of a file together with a lifetime of the reservation and estimated file size is delivered. The file state remains PERMANENT. The following possibilities arise:

- Cancel of the file transfer – all the reservations are canceled, file returns to initial state – PERMANENT;
- Confirmation of the transfer ending – Data Container checks, whether the file has been actually put on a server (i.e. at least one byte is written). In that case the file goes into a PERMANENT state with a new location, in opposite case it stays in a PERMANENT state without new location;
- No reaction – Data Container automatically checks the transfer state after the lifetime of the reservation. If there are no locks on the file at the moment (i.e. it's not being transferred at the moment) the operation is finished. Data Container checks, whether the file has been actually copied on a server (i.e. at least one byte is written). The file goes into a PERMANENT state.

Duplicating a file

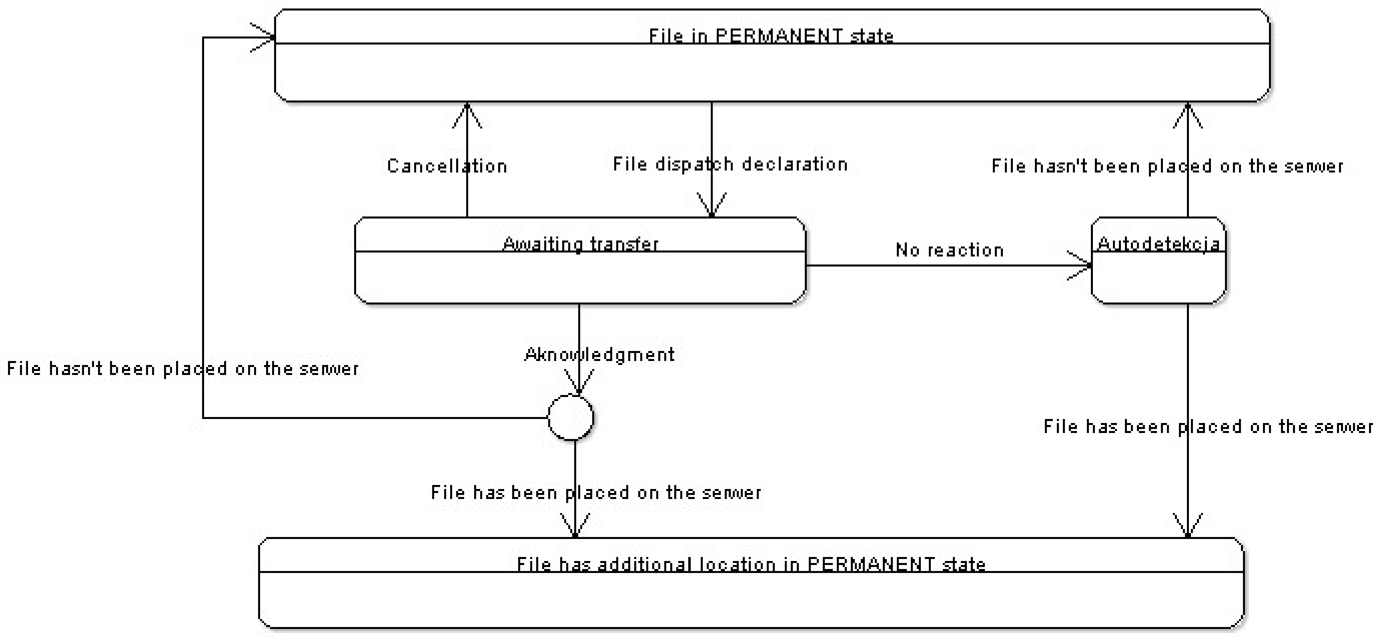
File in PERMANENT state
Cancellation
File dispatch declaration
File hasn't been placed on the server
Awaiting transfer
No reaction
Autodetekcja
Aknowledgment
File hasn't been placed on the server
File has been placed on the server
File has been placed on the server
File has additional location in PERMANENT state

Figure 6 File states during duplicating operation

Initially the file is in a PERMANENT state. A request to create a physical instance of a file together with a lifetime of the reservation and estimated file size is delivered. The file state goes into an IN_TRANSFER state. The following possibilities arise:

- Cancel of the file transfer – all the reservations are canceled, file returns to initial state – PERMANENT;
- Confirmation of the transfer ending – Data Container checks, whether the file has been actually put on a server (i.e. at least one byte is written if nonzero size is declared) and whether the file is accessible (whether the file is still used by process uploading the user data). In that case the file goes into PERMANENT state with a new location, in opposite case it stays in a PERMANENT state without new location;
- No reaction – Data Container automatically checks the transfer state after the lifetime of the reservation. If there are no locks on the file at the moment (i.e. it's not being copied at the moment or, in case when the declared size is 0, whether the file is unused) the operation is finished. Data Container checks, whether the file has been actually transferred on a server (i.e. at least one byte is written). If yes, the file goes into a PERMANENT state with updated location and size. Otherwise, it returns to the initial state.

DMS interfaces

This chapter presents each DMS module interface. Description of these interfaces is useful when an external application that uses DMS system and its modules are prepared. In that case the most important for developer module interfaces are: the Data Broker interface, the Metadata Repository interface, the Data Container module interface, the SRS Container proxy interface and the Policy Broker interface. All of these are described below.

Data Broker interface

Data Broker delivers services for managing and serving data, managing data logical structure and external data banks access. All their services are exported through the Web Service technology basing on the Axis implementation. Data broker is modular set of components which are:

- Data Management – the primary module delivering data manipulating operations on files and directories;
- Metadata Management - enhances the basic functionality with the possibility of adding metadescriptions and performing metadata-based exploring;
- Proxy Access module - delivers services for accessing external data sources which are not integrated with the DMS resources.
- Policy Management - it consists of the services for data access restriction (Security), data access policy management (Access policy) and user and group information management (Identity Management).

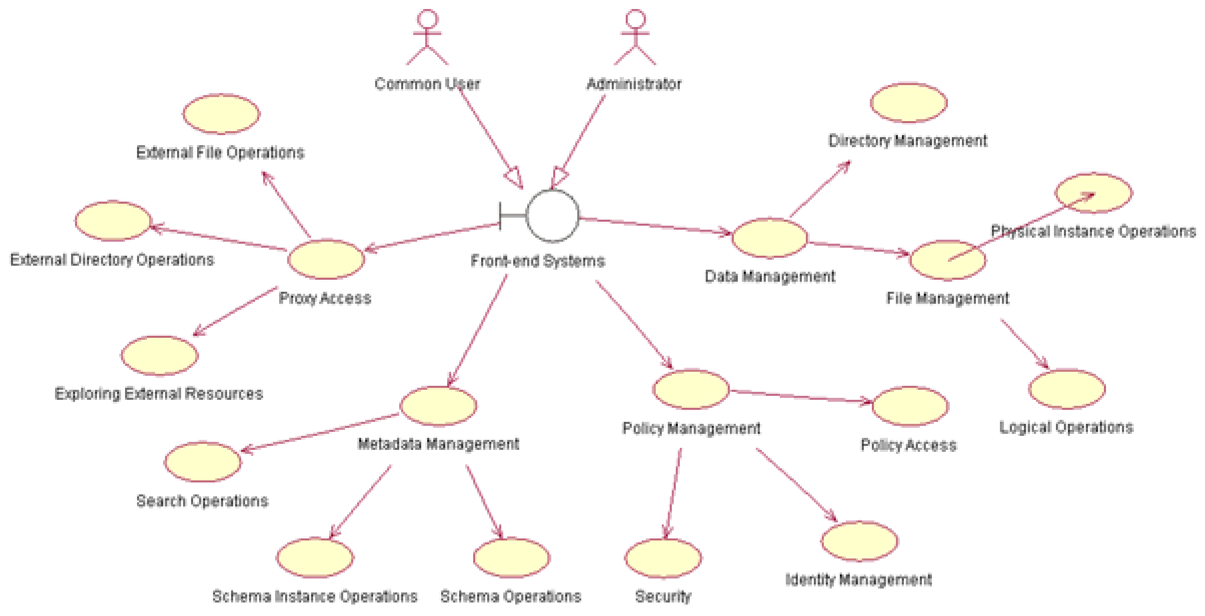

Figure 7 Data Broker public interface

DMS Data Broker delivers the integrated set of services that can be divided into following functional groups:

- Data Management Services:
    - Directory Structure Management - allowing to manipulate on meta-directories;
    - Data File Management - delivering operations on the logical file instances like adding, renaming file (Logical Operations) and physical file manipulation services as adding physical data file instance or removing physical file instance (Physical Instance Operations).
- Metadata Management Services:
    - Schema Operations - for adding schema or retrieving schema attributes;

    - Schema Instance Operations - schema instance is defined as schema assigned to the DMS object (a file element or a directory node) with specific values of the schema attributes
    - Search Operations - for searching through DMS resources basing on the schema attribute values of the schema instances assigned to the DMS objects.

- External Database Access Services:
    - External Directory Operations - to read external directories contents. As it is defined in example of the SRS data bank access it allows to retrieve the data banks list grouping data sets. Such databank is defined as logically separated libraries of data entries stored in flat files containing separate entries or groups of entries;
    - External File Operations - a set of operations for managing data entries from the databanks (retrieving single data or data sets);
    - Exploring External Resources - searching through external data sets basing on the metadata schema describing them. SRS Proxy bases on the metadata attributes describing the data entries. During the query building, user specifies fields to search and sets values for them. The returned list of entries consists of these records, which fulfills query conditions. The search engine has to be prepared to search through databases with different metadata attributes describing entries, depending on the internal databank structure.
- Policy Management Services:
    - Security aspect - mechanisms for protecting resource access. It includes the authentication mechanism and access authorization.
    - Identity Management – users' identification and their classification to groups of users. User groups allow defining a common access policy for a set of users and also simplify the procedure for the definition of the resource access policy.
    - Policy Access - a set of functions allowing the creation of the policy for resource access authorization. It delivers a possibility of creating operation definition and resource definition objects and giving the privileges to users or user groups on DMS objects (as resource, service and operations).

The data management interface

The following figure presents the set of DMS services collected to the data access operations group. This group of functions conforms to the `ISOAPDataBroker` interface.

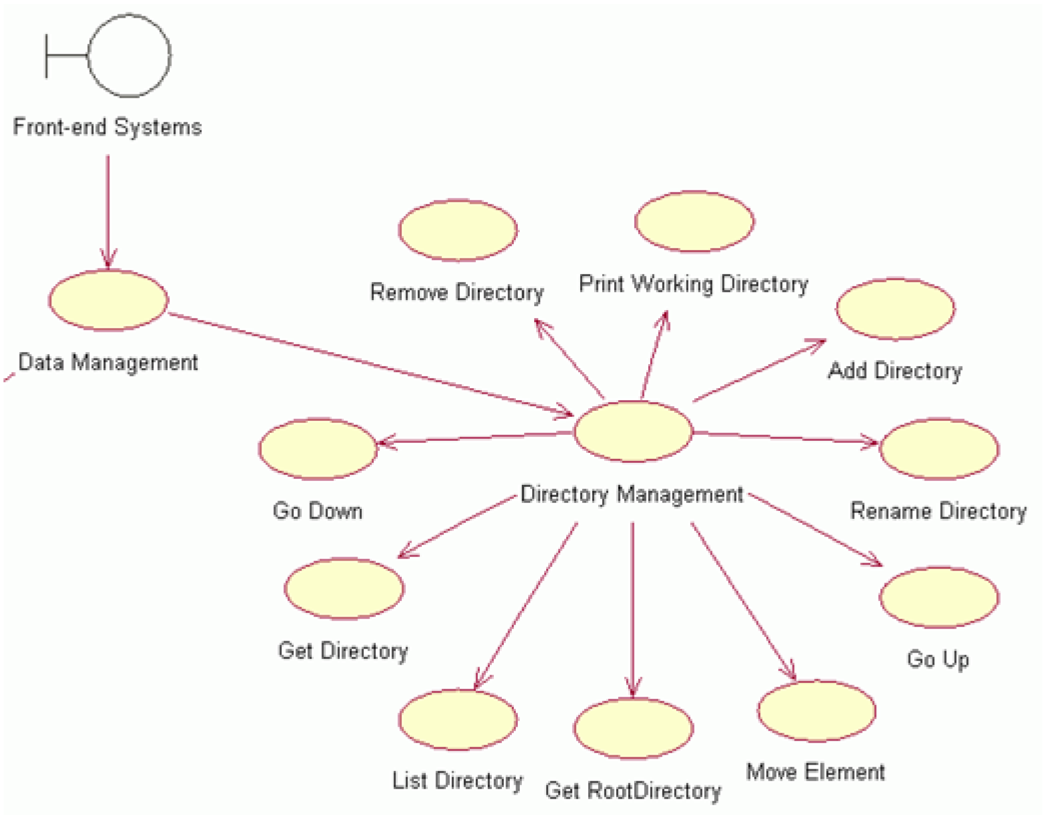

Figure 8 Meta data directory management services

The following diagram presents the services structure for managing data files. Please refer to the Java documentation for the use case description and further information.

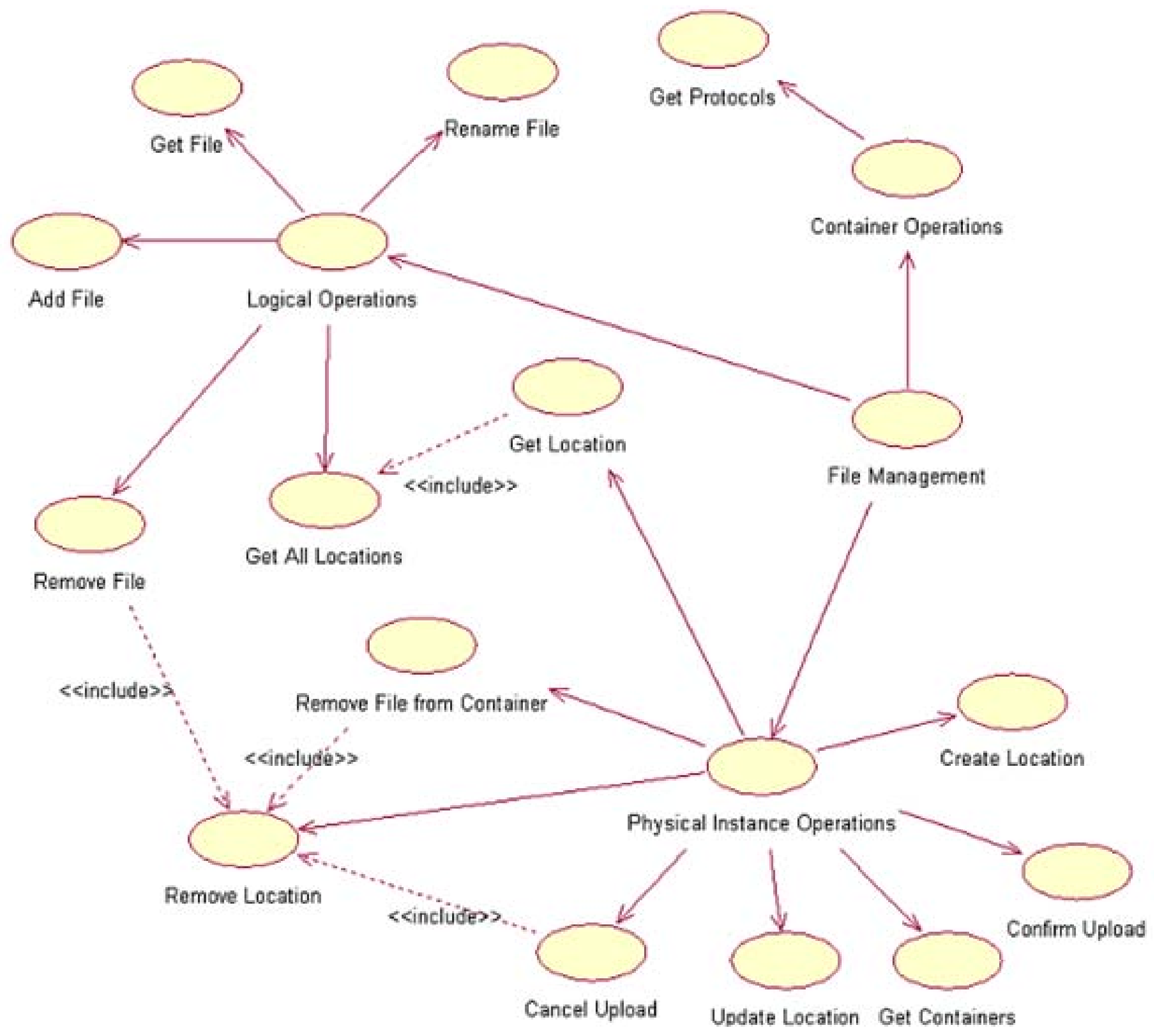

Figure 9 Meta data file managing

The metadata management interface

The metadata management interface is an extension of the Data Broker interface with the possibility of performing metadata search and retrieving metainformation about DMS objects.

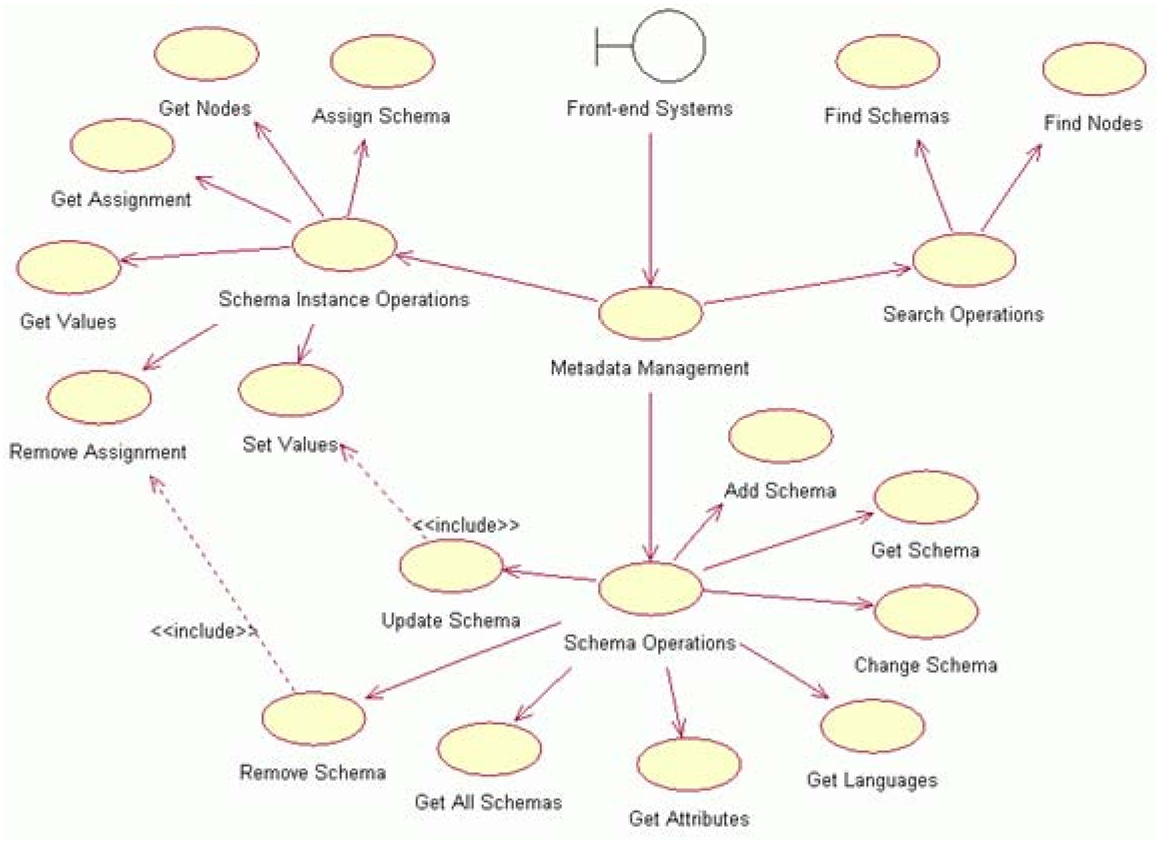

Figure 10 Data broker metadata services

Functional meaning of these use cases fulfils the interface methods of the Data Broker (`ISOAPMetadataBroker`) described in the Java documentation (JavaDoc). Please refer to the Java documentation for the use case description and further information.

The interface for external databases

External databases interface provides a unified set of services for managing external resources with hierarchical structure. These resources can be described with an additional layer of the metadata. These metadecryption makes it possible to data exploration and perform search. At the pilot installation of the DMS environment it is enabled access to the data sets of the SRS system (biological set of databases) with following functionality:

- exploring the SRS databank;
- data access proxy;
- executing the queries enabling to search through the SRS content according to the user defined conditions.

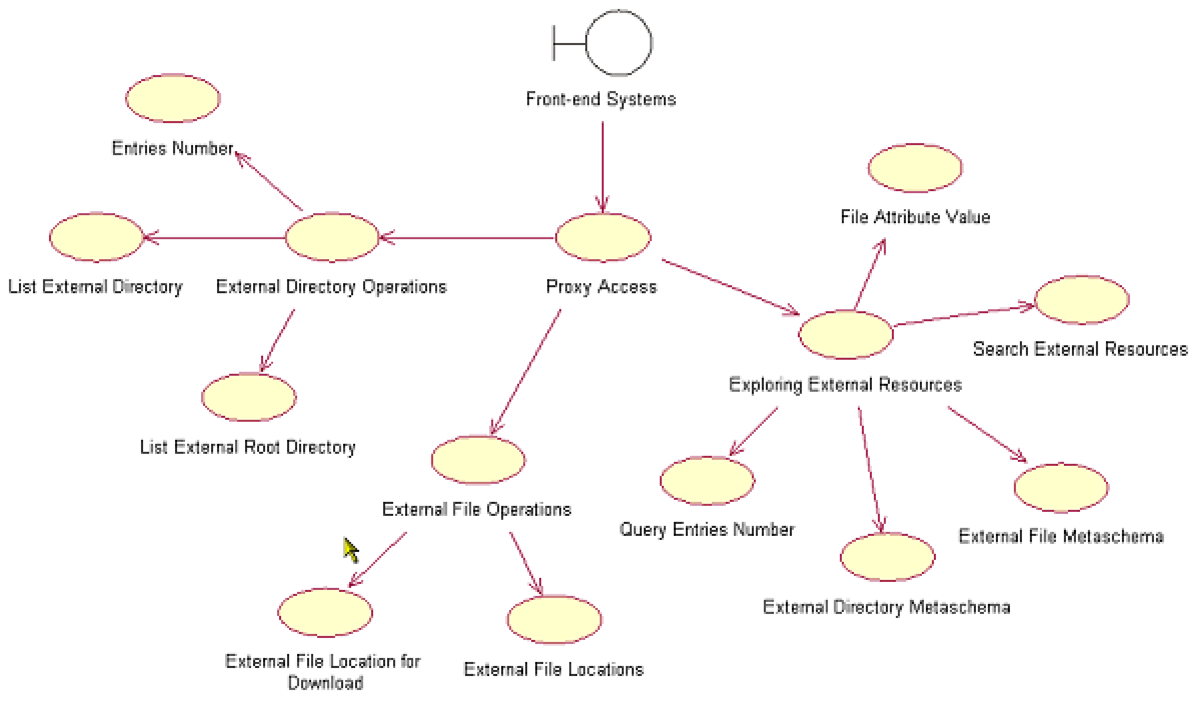

Figure 11 External databases services (Proxy services)

Functional meaning of these use cases conforms to the interface methods of the Data Broker (`ISOAPProxyBroker`) described in the Java documentation (JavaDoc). Please refer to the Java documentation for their detailed description and further information.

The Policy interface

Policy management services are delivered in a common interface creating comprehensive set of functionality for defining a security policy and protection of DMS resources. It covers a set of services for widely mean system security which constitutes identification mechanisms, authorization mechanisms, identity managing services and definition of data access policy services for internal resources, DMS modules and system operations.

According to the assumptions for DMS system, each user who sends a request has to pass out the identity information (basing on login procedure) which allows to authorize a his/her access to the DMS resources and operations. It is possible to login directly i.e. using standard login procedure with user name/password or identify through the user certificates set on the connection layer, what is a recommended option.

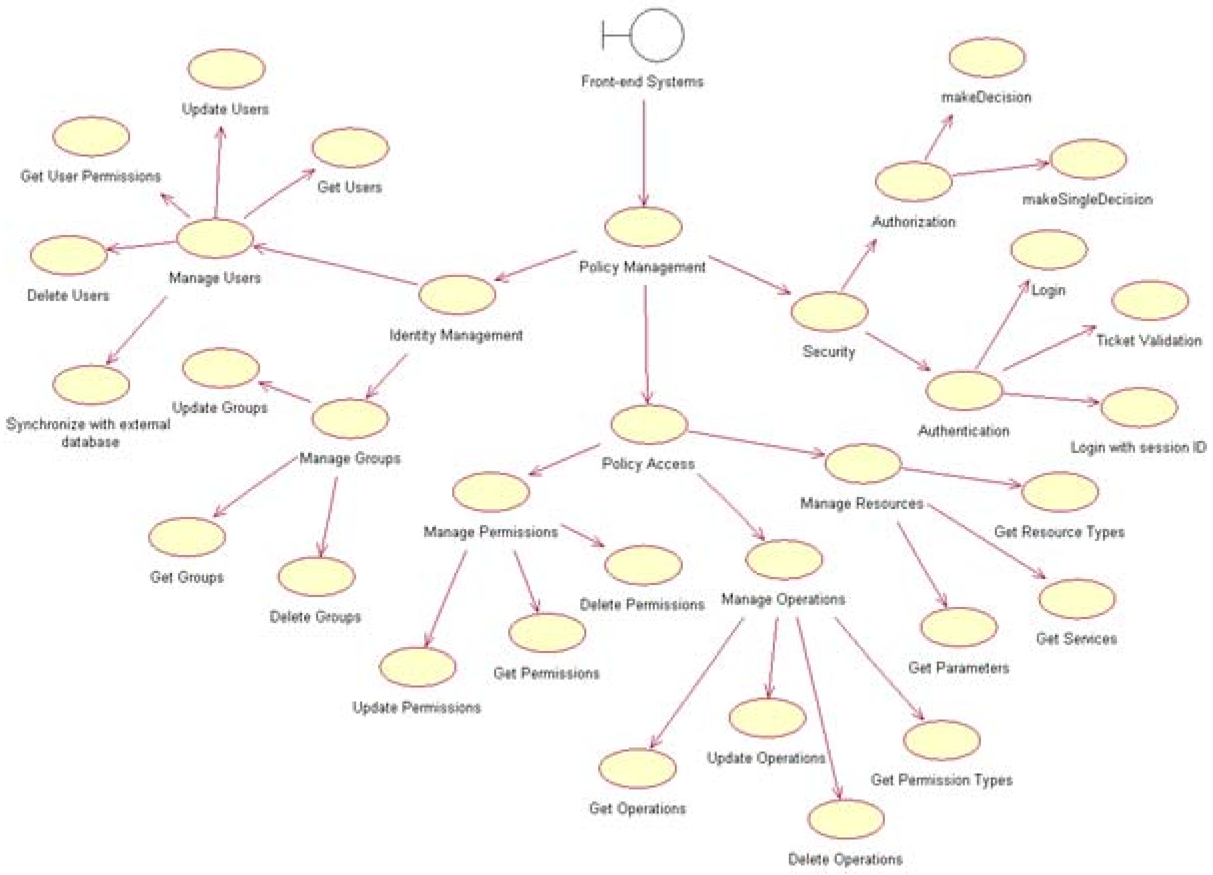

Figure 12 Policy services

Functional meaning of these use cases conforms the interface methods of the Data Broker (`ISOAPPolicyBroker`) described in the Java documentation (JavaDoc). Please refer to the Java documentation for the use case description and further information.

Metadata Repository interface

DMS Repository is a central system module. It is the knowledge base for any taking decision in the data management environment. Repository is a communication point which provides services for other DMS modules as Data Broker and Data Container. It exposes a separate interfaces with a group of services for both these modules.

The interface for the Data Broker

Metadata Repository provides following services for the Data Broker module:

- Node manipulation services allowing to manage the data delivered by the Data Broker through its interface
- Metadata managing services allowing to process the metadata description delivered by the Data Broker in its interface
- Registration services allowing to register Data Brokers in the Metadata Repository

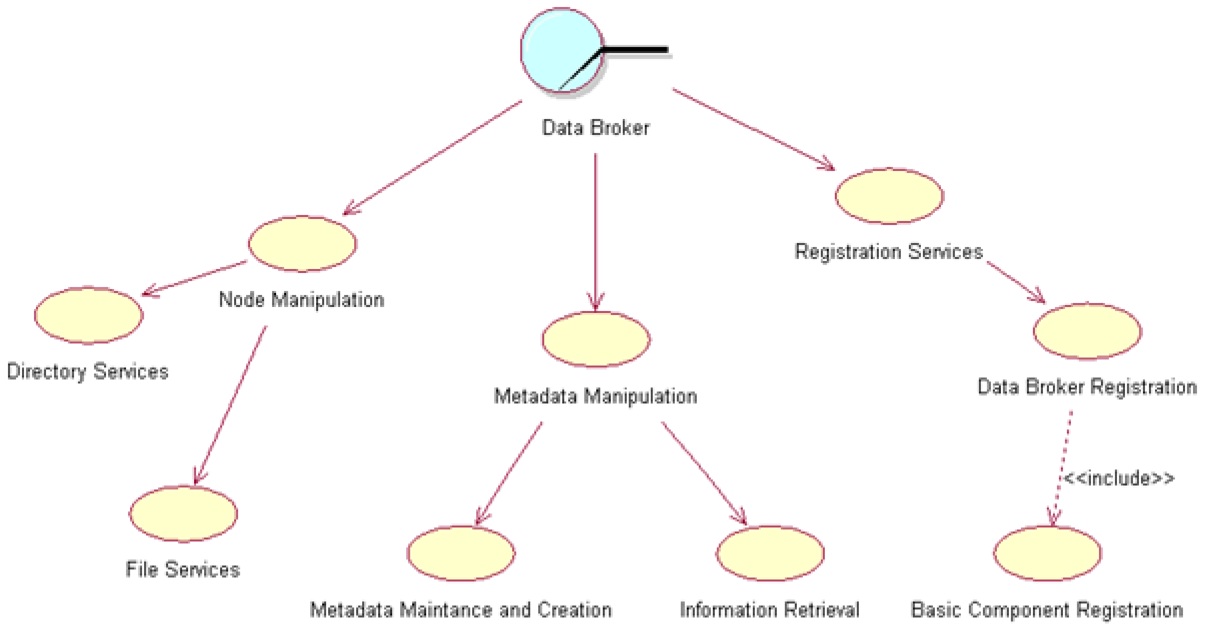

Figure 13 Data Repository for Data Broker interface

This set of functionality is well described in the technical documentation of the Repository module (JavaDoc API). The main rule which was taken during the project phase assumes that each of the Data Broker interface method has a corresponding method in the Repository interface, usually with the identical name.

The interface for the Data Container

The Data Container uses Repository interface methods to register itself (during system startup) and notify about user data set transfer to/from the data server. On the basis of the information Repository dynamically reacts to the changing of data files states and updates the knowledge base about file locations.

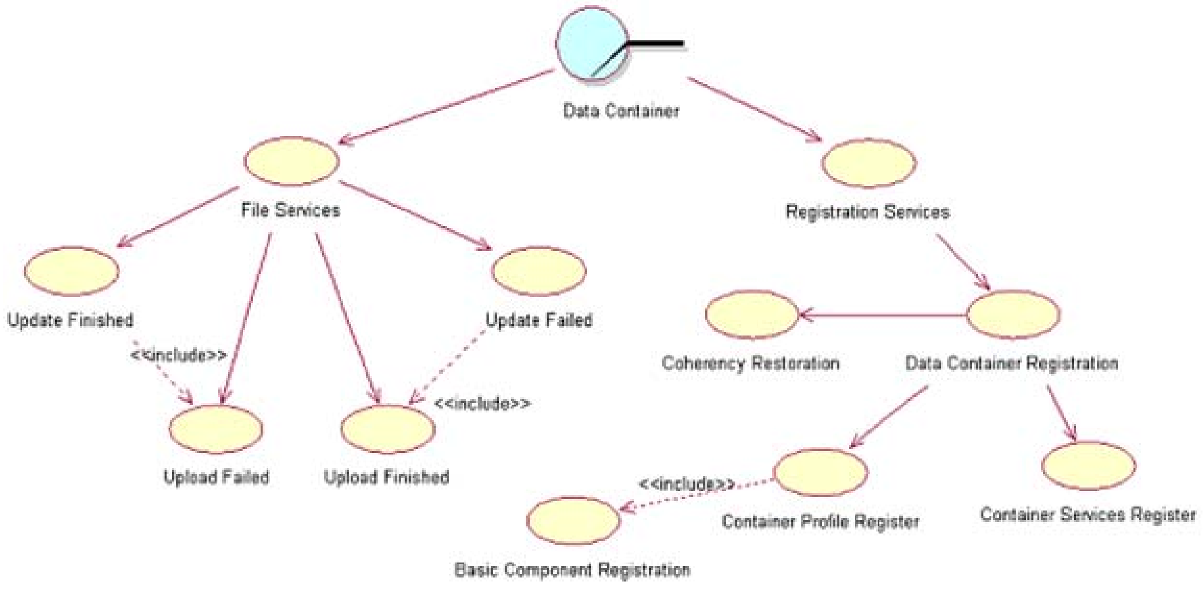

Figure 14 Data Repository for Data Container interface

Data Repository provides following set of services for the Data Container:

- Notification about data transfer finishing - method which allows to inform the Repository about adding new instances of the user data file to the Data Container or about update of the data set content (Upload Finished);
- Notification about data transfer canceling - service used by Data Container when adding new data file instance was interrupted (Upload Failed) or when update of data content was interrupted (Update Failed). It informs the Data Repository about failure of the user data file transfer.
- Data Container registration - a set of the services allowing to register the Data Container in the DMS environment, notify about actually active data transfer servers and their configuration and allowing to perform the data knowledge integration between the Data Repository records and Data Container.

All these services are described in details in the JavaDoc API for Data Repository technical documentation.

Data Container interface

Data Container provides a set of services for the Metadata Repository which are required to perform its resources managing. The communication between the DMS Repository and the Data Container is described with the designed interface common for different instances of data storage module. These instance may vary according to the different type of used resources (file system, data archiving system basing on tapes and database or external publicly available information source, like SRS system), and with the technology used to prepare the module. There is only one main condition which have to be fulfilled - all these modules have to derive the common interface and functionality covered by it. Similar to other DMS modules, Data Container communicates with all other components basing on the SOAP protocol. Its Web Services are used by the Repository module to handle the data access user requests.

The interface for the Metadata Repository

Data Container delivers a set of services for control the resource access by the Repository module. Data sets stored with use of Container are available for users after its appropriate preparing. Preparation of this data is performed on a user request sent to the controlling whole process Repository. The Data Container services may be divided into following groups:

- Access services for Container resource - creating data reservation (performed before data file is sent to the container), initiating the data update, performing the data file downloading and deleting data sets from the container resources;
- information services for Container - getting the size of the available space in the container resources, the size of the space used by user data files, the size of the overall space in the container resource and the reserved container space for future transfer;
- A set of administrative services for container module initiating, enabling/disabling data transfer server, checking a file or a persistence of reservation.

You can find more detailed information connected with Data Container module functionality in the JavaDoc technical documentation.

The interface for a user application

Data stored in the Data Containers modules are available for user applications with the use of standard data transfer servers. According to the configuration, Data Container may cooperate with the FTP, GASS or GSIFTP server. User application sends a data access request to the Data Repository which returns the data URL location address to the container resources storing required data files. Further communication between users application and data storage module is performed with the use of transfer server operating under the URL address. It makes the container resources visible for the world through the configured data access servers according to the holding protocols. Data Container Web Services are used only by the Data Repository to initialize the data access on the user requests.

SRS Container

SRS Data Container delivers data controlled by external biological database called SRS. In fact it collects the functionality of the external data repository (by delivering services for getting listing of the database contents, retrieving data objects and metadata connected with them) and Data Container (serving data access using GASS server or FTP, GSIFTP, HTTP and HTTPS protocols) in a way similar to the standard DMS Data Container module. Because of the specific nature of the SRS external database (read only data access mode) and necessity of the SRS content delivering, SRS Data Container implements its own interface which differs from the standard Data Container interface.

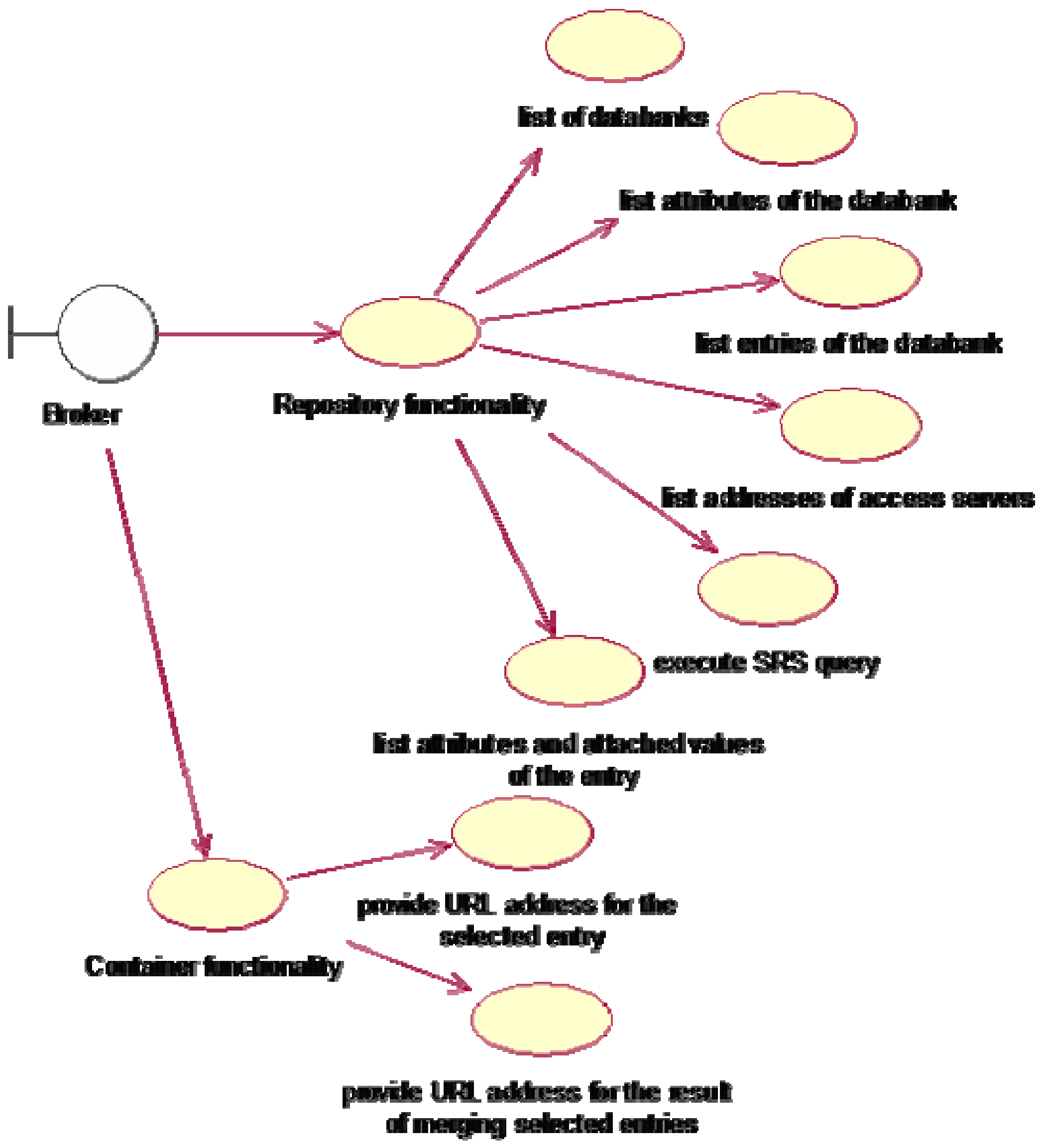
list of databanks
list attributes of the databank
list entries of the databank
list addresses of access servers
execute SRS query
list attributes and attached values of the entry
Broker
Repository functionality
Container functionality
provide URL address for the selected entry
provide URL address for the result of merging selected entries

Figure 15 SRS Data Container interface

The interface for the Data Broker

SRS Data Container provides following services for the Data Broker:

- Resource access services - services allowing to retrieve data from the SRS and share them using the chosen protocol. It is possible to get single data entry of the database library and the set of query result entries merged into one data file;
- Data access procedures - services for retrieving of the SRS contents. It is possible to retrieve the listing of databanks library, listing of the single databank, retrieving the metadata description of the data stored from the databank, the databank data entries and performing query in the data banks resources.

The following service methods are available for the SRS Data Container:

- Getting the databank element (lock data for reading) - a method for retrieving databank entries content under request protocol, time interval and data format;
- Getting data merged into data set - method for retrieving merged data file containing set of data entry files being result of user query. Merged file is available under request protocol, time interval and in requested data format;
- Retrieving databanks list;
- Retrieving databanks entries list;
- Retrieving databanks entries list in extended mode, that is with a possibility of paging the result;
- Retrieving the number of databanks elements;
- Retrieving the list of available transfer servers (protocols) on the SRS Data Container;
- Retrieving databanks attributes (SRS library fields);
- Retrieving the entry single attribute and its value for single data entry;
- Getting the list of attributes of the databanks elements and their values (SRS entry fields value) in a raw format;
- Performing query on the SRS resources according to the user defined criteria;
- Retrieving the number of entries being result set of the query on the SRS resources according to the user defined criteria;

The Data Container module functionality is also well described in the JavaDoc technical documentation.

The DMS use cases

The use cases described in this point allow introducing the basic principles of the DMS use. Each of this section presents hypothetical schema of the user activity in the DMS. This schema presents an example of typical actions taken by the user. These actions are: logging into the system, creating a new data file (with its physical location), adding a metadata schema and using this schema to describe data file.

Basic operations on a data file

This section describes basic schema of user activity when data file is uploaded or retrieved. This schema usually consists of the login operation, retrieving user's home directory, changing path and uploading data. Additionally this section describes some basic metadiretory operation as creating metaschema and adding metadescription to the data file.

Login

This point describes the login procedure using one of the following methods:

- `AuthorizationStatus loginGSI();`

- `AuthorizationStatus login(String, String, String);`
- `AuthorizationStatus loginSession (String).`

According to the DMS configuration and the character of the subject using DMS there exist separate login procedures for login basing on user certificates (paramless `loginGSI` procedure), user name - password - domain triple or login method basing on the session identifier. The result object of the login method called `AuthorizationStatus`. If the DMS authorization process finishes with success (check the `AuthorizationStatus.getOperationStatus`) then the `AuthorizationStatus` contains the user identifier (`AuthorizationStatus. getUserID()`), user time ticked (`AuthorizationStatus.GetTicket()`, not used when certificate based user authentication is used) and the identifier of the user home directory (`AuthorizationStatus.getUserHomeDirectoryID`).

Notes

- The user home directory identifier is not valid for users from DMS_SEC_AUTHORITY domain. This is a distinguished method which indicates users not having the home directory. Users from this domain may use the system root directory to start the data structure exploring (by the Data Broker's `getSystemRootDirectory` method).
- The user autenthication methods used in the DMS are based on the user certificates (certificates are sent in the protocol layer) or on the user tickets (user uses ticket got after login), according to the DMS configuration. Both these authentication methods are described in the DMS Admin Guide in the Heimdall module description section.

Retrieving the user's home directory

Listing of the user home directory is possible by the following method:

- `Vector listUserDirectory(int , String)`

This method allows to retrieve the listing (directory contents) of every directory to which user has appropriate privilege. This method is expecting the identifier of the directory to list (e.g. user home directory from `Vector listUserDirectory(int , String)`). The second parameter represents a time ticked (also available after user login from `AuthorizationStatus.GetTicket()` method).

Notes

- The time ticked is not used in case of user certificate based authentication. Time ticked may be null in that case or may be used a real one - got from the `AuthorizationStatus.GetTicket()`.
- The certificate-based authentication allows to use the DMS services without direct login. User is authenticated and access is authorized on each DMS call on the base of his/her certificate.

Adding a datafile

Adding a data file is possible with use of the following method:

- `FileElement AddUserFile(String, int, String).`

This method allows to pass the file name string and the identifier of the file parent directory element. The parent directory identifier may reference to any valid directory from the directory structure to which user is able to write (has appropriate privilege). In most cases it will be the user home directory, which identifier is available from the `AuthorizationStatus.getUserHomeDirectoryID`. The third parameter of this method is a user time ticked. As a result it is returned the `FileElement` object representing metadescription of the new file.

Notes

- Please keep in mind that this method creates a logical instance of the new file as a DMS Repository entry.
- The user certificate based authentication allows to use the DMS services without prior login. User is authenticated and access is authorized on each DMS call basing on user certificate.

Creating a file location

To create a physical file location for the logical file created in the previous step it is possible to use following methods:

- `URL makeUserFileLocation(int, long, long, boolean, String, String)`;
- `URL updateUserFileLocation(int, long, long, boolean, String, String)`.

The `updateUserFileLocation` method is an extension of the `makeUserFileLocation` method - it allows to overwrite the file content existing on the server. It has to be mentioned however that when the file is replicated, using the `updateUserFileLocation` is interrupted and an exception is thrown. The result of these two method is a URL address identifying the physical destination location of the file to upload. User uses this address to initiate real data transfer.

Notes

- The certificate-based authentication allows to use the DMS services without prior login. User is authenticated and access is authorized on each DMS call basing on a user certificate.

Data transfer confirmation

After data transfer finishes, the DMS can be informed about it by using method:

- `boolean confirmUserFileUpload(int, URL, String)`

This functionality helps to inform DMS Repository immediately abut user data transfer finishing. It is used on special need when Repository changes have to be visible right the operation ends. The `confirmUserFileUpload` methods expects the identifier of the data file and URL address of the physical location of the transferred data. Both those parameter are available as return results of the previously described `AddUserFile` and `makeUserFileLocation`/`updateUserFileLocation` methods.

The `confirmUserFileUpload` returns logical value TRUE if DMS detects a data transfer finish and updates the Repository state about data file state.

Notes

- Please keep in mind that DMS automatically updates the file state in the Repository node after detecting file transfer. The finish data transfer detection is performed just after the time interval of the reservation has been exceed. This time is given during the location creation. Because the reservation time may be apparently too long, this functionality allows to inform Repository immediately after data transfer finishes.
- The certificate-based authentication allows to use the DMS services without prior login. User is authenticated and an access is allowed on each DMS call basing on user certificate.

Adding a metadata schema

Adding metadata schema is possible with the use of the following method:

- `Schema addMetaSchema(Schema, Attributes, String);`

Adding metadata schema requires to specifying the schema parameters as its name or description and list of optional attributes. These parameters have to be passed to the `addMetaSchema` method in fields of specially designed objects called `Schema` and `Attributes`. After proper finish user gets as a return the `Schema` object with the schema identifier.

The certificate-based authentication allows to use the DMS services without prior login. User is authenticated and an access is allowed on each DMS call basing on user certificate.

Assigning a schema to the data file

The previously created metadata schema (please see above) may be assigned to the data file using following method:

- `SchemaInstance assignMetaSchemaToFile (Schema, AttributesInstance, int, String);`

Having metadata schema it is possible to assign them to the data file or directory node objects. To describe data file with the metaschema it is necessary to chose the file for describing (defined by the identifier) and set the schema attribute values. The array of values are passed to the method by the `AttributesInstance` object. After that step it is possible to assign the metaschema to the data file object by using one of the available `assignMetaSchemaToFile` method version.

DMS and Security

DMS is based on web services deployed on Axis. The WS may be use secure transport layer (HTTPS) with possible of using Globus user certificates for user authentication and access authorization basing on it. Access to the DMS resources is controlled using the Policy Broker module basing on Heimdall internal module. User access authorization may be configured by authorization plugins. The open character of the software allows preparing plugins according to the users' needs. At the highest level of security it is possible to use Globus user certificates with authorization basing on GAS (Grid Authorization Services) plugin.

The DMS is developed with a special care on the security aspects. These were taken into consideration at different levels:

- the DMS web services may be configured to use with the secure access protocol (SSL connections);
- the data file transfer to/from the DMS Data Containers may be realized by the secure servers handling FTPS or GSIFTP protocols;
- users may be authenticated basing on user name/password/domain or user certificates (according to the chosen configuration);
- resource access is realized basing on internal policy with possibility of user authorization via external plugins;

3. Metadata Repository

The Metadata Repository is a registry for metadata specific for knowledge discovery tools and other services. This repository manages metadata to support the following main tasks:

- Publish (register) tools by service providers.
- Find (discover) tools by service clients.
- Bind (invoke) tools by service clients
- Modify existing tool metadata.
- Retrieve all tool metadata (for metadata browsing tools).

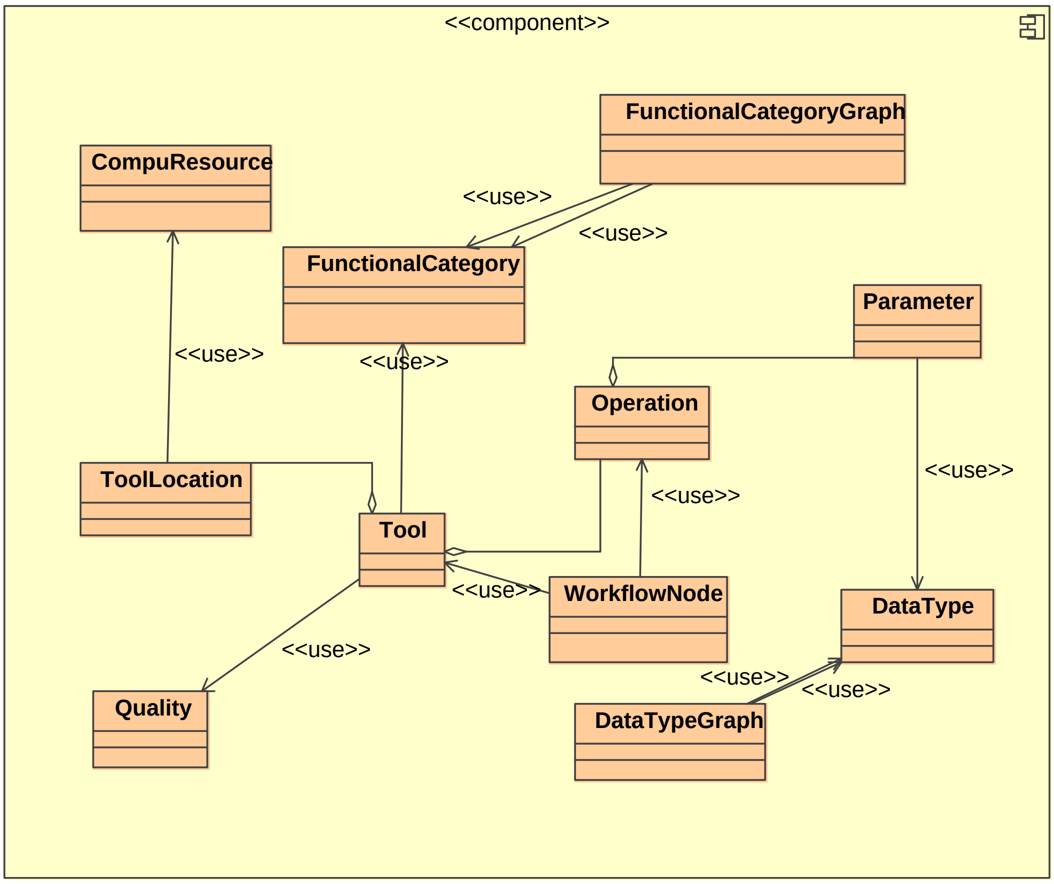

Figure 16 Overview of the metadata entities and their relations

The repository's schema includes information that is potentially interesting for tool discovery and invocation (tool description in Figure 16). The domain of the metadata is services (tools), operations (specific functions of a tool), workflows (pre-defined flow of data between several operations), functional categories (descriptions of tool functionality) and data-types (the input or output data type of operation parameters).

The complete description of the metadata repository, its design, and its services and APIs are included in the Deliverable 6.3 [D6.3].

4. Conclusions

Data in ACGT are managed in various places depending on their nature and role in the system. We have identified the following entities that:

- Grid Data Management Services: The Grid data layer is responsible for the maintaining the user uploaded data or data sets that either lack structure, are too cohesive and self contained to be indexed by the ACGT databases, or are just too large to be handled elsewhere. The DMS infrastructure offers location independent identification of the data files and efficient storage and transfer. Furthermore relevant metadata can be attached to the data files and managed by the same services so that synchronization between the data and the metadata is easier.
- ACGT Database Layer: The domain specific databases are in the authority of the relevant VOs. The Semantic Mediator offers a higher level data access interface that unifies these databases and offers cross database queries with correlation and joins.
- ACGT Metadata Repository: The service and workflow related metadata information is stored in this metadata registry that supports the high level integration of the analysis tasks into scientific workflows. The functionality, input/output parameters and data types, location and mirroring information in cases where multiple replicas of the service exist, and so on are maintained there. The metadata repository expose retrieval, discovery, and update functionality through its APIs and unique identification of the stored information is supported by the adoption of URI schemes for the ids.
- Workflow Repositories: The Workflow related repositories store metadata related to the "cause-and-effect" process followed in the workflow enactments. In particular they store provenance information that relates the data to the service or tool that generated it and even further to the initial input data and parameters used.

There are a lot of open questions and issues to be discussed. For example the DMS handling of metadata remains to be evaluated because a restricting design choice allows only the registering of attribute keys and values and not more rich structures. Also the design of the Workflow Repositories for registering the provenance information is something to be decided in the near future so that the gap between static content stored in DMS and the ACGT databases and the "actionable" content of the Metadata repository is filled in. This document therefore will be revisited and complemented as the development of the different components proceeds in the course of the project.

5. References

[D11.2] Deliverable 11.2 "Implementation of the ACGT core security services & initial implementation of the Pseudonymisation tool"

[D6.3] Deliverable 6.3 "Demonstration and report of a repository of knowledge-discovery-related metadata"

[SAWSDL] Semantic Annotations for WSDL and XML Schema, W3C Recommendation 28 August 2007, http://www.w3.org/TR/sawsdl/

[GRDDL] Gleaning Resource Descriptions from Dialects of Languages (GRDDL), W3C Recommendation 11 September 2007, http://www.w3.org/TR/grddl/

[RDFA] RDFa Primer: Embedding Structured Data in Web Pages, W3C Working Draft 17 March 2008, http://www.w3.org/TR/xhtml-rdfa-primer/