

---

# AstroGrid-D

Deliverable

---



## AstroGrid-D Information Service Requirements Specification and Architectural Design<sup>1</sup>

Deliverable	WG2-D1
Authors	Working Group 2
Editors	Mikael Höggqvist, Thomas Röblitz
Date	May 15, 2007
Document Version	1.0.1
Current Version	1.0.1
Previous Versions	0.1.0, 0.2.0, 0.2.1, 1.0.0

### **A: Status of this Document**

Public release.

### **B: Reference to project plan**

Deliverable D2.1: *AstroGrid-D Information Service*.

---

<sup>1</sup>This work is part of the D-Grid initiative and is funded by the German Federal Ministry of Education and Research (BMBF).

**C: Abstract**

The purpose of an AstroGrid-D Information Service is to provide storage and discovery mechanisms for metadata fulfilling the heterogenous resource and user requirements that forms AstroGrid-D. More specifically, metadata in AstroGrid-D consist of resource information, scientific annotations, user application and grid-service metadata. By storing this metadata in the Information Service, users should for instance be able to find geographically distributed datasets, locate their current jobs, publish their research metadata or see the current status of robotic telescopes.

The architectural design takes several requirements into account such as extensible schemas, easy metadata extraction for collaborations and fine-grained security. A simple interface for storage management and metadata discovery, hiding a more complex distributed storage back-end is envisioned.

**D: Change History**

<b>Version</b>	<b>Date</b>	<b>Name</b>	<b>Brief summary</b>
0.1.0	10.03.2006	Mikael Högqvist, Thomas Röblitz	Initial version
0.2.0	05.04.2006	Mikael Högqvist, Thomas Röblitz	Re-iterated the use case section and the architecture section. Incorporated feedback and other contributions. Adapted to the deliverable template.
0.2.1	08.05.2006	Mikael Högqvist, Thomas Röblitz	Minor changes. Language, added more information to the use case section. Moved scenarios to its own chapter. Referenced the requirements in the architecture chapter.
1.0.0	27.07.2006	Mikael Högqvist, Thomas Röblitz	Minor changes and polishing for the first public release.
1.0.1	15.05.2007	Mikael Högqvist	Typo and references.

**E:**

# Contents

Abstract . . . . .	2
Change History . . . . .	3
<b>1 Introduction</b>	<b>7</b>
1.1 Scope . . . . .	8
1.2 Document Structure . . . . .	8
<b>2 Use Cases</b>	<b>9</b>
2.1 Simulations . . . . .	9
2.2 Analyzation of data sets . . . . .	11
2.3 The Planck Process Coordinator (UC13) . . . . .	14
2.4 Integration of Robotic Telescopes . . . . .	15
<b>3 Requirements on the Information Service</b>	<b>16</b>
3.1 Grid and Generic Information Service Requirements . . . . .	16
3.2 User Requirements . . . . .	16
<b>4 Related Work</b>	<b>18</b>
4.1 Metadata storage . . . . .	18
4.2 Grid information services . . . . .	19
<b>5 AstroGrid-D metadata</b>	<b>21</b>
5.1 Metadata classes . . . . .	21
5.2 Metadata overview . . . . .	22
5.2.1 Resource specific metadata . . . . .	22
5.2.2 Integration of Robotic Telescopes . . . . .	23
5.3 Representation of Metadata . . . . .	26
5.4 Discovery . . . . .	27
5.5 Metadata schemes . . . . .	28
<b>6 Architecture</b>	<b>31</b>
6.1 Architecture Overview . . . . .	31
6.2 The AstroGrid-D Information Service . . . . .	33
6.2.1 Design objectives . . . . .	33
6.2.2 Architecture components . . . . .	34
6.2.3 Storage mechanisms . . . . .	37
6.2.4 Summary . . . . .	40
6.3 Security Model . . . . .	40
6.3.1 Access Control Lists . . . . .	41
6.3.2 Evaluating an ACL . . . . .	41
6.3.3 Security Levels . . . . .	42

---

<b>7</b>	<b>Interfaces</b>	<b>43</b>
7.1	Interface description . . . . .	43
7.2	Datatypes . . . . .	43
7.3	Exceptions . . . . .	44
<b>8</b>	<b>Scenarios</b>	<b>49</b>
8.1	Register a new grid job . . . . .	49
8.2	Find all running grid jobs for a user . . . . .	49
8.3	Access to application generated metadata . . . . .	50
8.4	Apply security information to metadata . . . . .	51
8.5	Update dynamic resource information . . . . .	51
	References . . . . .	53

# 1 Introduction

The AstroGrid-D metadata management service is a core component of the astrophysics community grid middleware. It aims at storing information about diverse entities such as compute resources, robotic telescopes, data sets, compute jobs, application-specific progress information, service states and service properties, etc. These entities are managed by different components of the AstroGrid-D middleware, for example access to data files is provided by the file management service (work package 3) or compute jobs are managed by the Grid-job management service (work package 5). Our approach to managing metadata decouples the handling of the metadata and the operations on the corresponding entities. Thus, the information is easier to access from other components, information on different entities can be correlated in a more flexible way and the metadata for different kinds of entities is managed with the same methods (e.g. for updating and querying information). The first characteristic is important to the portability of the AstroGrid-D middleware and supports the integration of third-party components used by other communities. The second one is crucial for complying with complex requirements such as executing data-intensive compute jobs near the storage of the input data. The last, in particular, is important to apply a common community-wide access rights handling model to metadata.

Conceptually, the metadata management service provides a global view of the state of all entities. However, this does not imply that only a single instance of the metadata management service serves the whole AstroGrid-D community. In this document we focus on the conceptual realization of the metadata management service rather than its implementation. The interfaces presented in Section 7 are the same regardless of the implementation.

The metadata management service provides methods for adding new information, updating or removing existing metadata and discovering information. The Information Service needs to be flexible enough to support the diverse classes of metadata derived from the typical scenarios in the astrophysics community. Because information may be sensitive and the service is storing all AstroGrid-D metadata, enhanced security mechanisms are required. Similarly to well-known mechanisms in network file systems, there will be three levels of access restrictions: user, group/VO and public. The access to any information can be restricted at the attribute level, i.e. while some attributes may be publicly available for reading, others may only be accessible for certain groups/VOs or only by the owner of the information. While it is straight-forward to define such restrictions with access control lists (ACL) special emphasis must be put on the performance of accesses when ACLs are used.

The requirements on the AstroGrid-D metadata management service were derived from use cases developed by the AstroGrid-D community. The use cases describe typical scenarios for running simulations, analyzing data and initiating observations on robotic telescopes. After gathering a first set of scenarios the architecture group developed a detailed questionnaire to ask for a more detailed description. This second step was also used to transform the existing use cases into a unique structure, thus allowing an easier analysis of the requirements of the

scenarios.

## 1.1 Scope

The Information Service will not:

- define the interfaces for other services that are using the generic Information Service interface.
- define the security policies for AstroGrid-D. It will support two security related schemes for manipulating entries in the Information Service storage.
- define interfaces for user and group management.
- define how the stored metadata is presented to the metadata consumer.

## 1.2 Document Structure

The remainder of this document is structured as follows. Section 2 provides an overview on the use cases. The requirements derived from these use cases are described in Section 3. In Section 4 we review existing tools and related work concerning the management of metadata. The following section presents the architecture for the AstroGrid-D metadata management service. The interface for accessing the service is defined in Section 7.

## 2 Use Cases

The members of the AstroGrid-D community have developed a couple of use cases representing typical astrophysical scenarios. Besides a precise description of the current way of use, the use cases also outline how to use a distributed environment, i.e. the AstroGrid-D community Grid.

The use cases were developed in two phases. In the first phase the users where asked to provide an overview of their application, this was mainly to allow a freedom of specifying current and eventual future needs. The second phase included a detailed questionnaire that ensured a more comprehensive view of each use case. We have classified the use cases into (A) simulations, (B) data analyzations and (C) astronomic observations. In the following paragraphs, for each class, we introduce the main characteristics concerning the management of information.

### 2.1 Simulations

There are six use cases describing simulation runs. A simulation usually expects a relatively small amount of input data, which is processed in an iterative manner. At each simulation step a large number of output data may be generated depending on the problem size and the configuration parameters. Because such simulations perform complex computations they are usually requiring long execution times. Thus, most of the applications already provide mechanisms for observing their progress. Some applications even facilitate on-line steering, e.g. adapting parameters to speed-up the processing.

The enhanced mechanisms for application monitoring and steering pose new requirements on the metadata management service. It must be able to cope with a wide variety of metadata schema which are not known a-priori. Also, the metadata management service must be able to protect the information such that only certain community members may access the application-specific information.

#### UC1 - NIRVANA

NIRVANA is a an AMR (Adaptive Mesh Refinement) computercode for solving magnetohydrodynamic equations and the Poisson equation.

The use case does not explicitly specify requirements on the information service. However, the usual Grid job and data file management operations associated with executing the scenario pose several requirements on an information service. The information service must store metadata about compute resources including their software environment, data storage resources, network resources, compute jobs and the locations of a data file. Besides the usual activities associated with running a job on the Grid the scenario requires access to job progress information in a log file. Parts of this information may be stored in an application-specific metadata object.

## UC2 - AMIGA

AMIGA is a code for cosmological N-body simulations, i.e. simulating the whole Universe in a computer.

The use case AMIGA is very similar to the NIRVANA use case wrt. the requirements related to job and data file management operations. Also, the scenario may require certain job progress information to be stored in the information service. In addition, the use case outlines that simulation results shall be made accessible to other scientists (astronomers). This may pose some requirements on the access control to both the metadata (describing the results) and the data results itself. The latter is handled by the data file management service. The former must be handled by the information service.

## UC3 - NBODY6++

NBODY6++ is a member of a family of high accuracy direct N-body integrators used for simulations of dense star clusters, galactic nuclei, and problems of planet formation.

While the NBODY6++ use case shares the core features as the previously discussed simulation use cases, it also exhibits some more specific properties. The application can use special hardware (GRAPE boards) to gain significant speed-ups. Because it uses a blocking communication pattern special attention must be paid to the balance of computation and communication performance of the resources to be used. Hence, the description of the compute resources must support the decision process on where and how many processors a job is executed. Although it is not explicitly mentioned in the use case the scenario might benefit from a data file management that allows to attach arbitrary information to a file, e.g. source code version number, creation date of a result file, etc. Furthermore the use case may require a mechanism for defining which user has access to monitoring data generated by a running application. This has to be considered if the monitoring data is stored in the information service. The users of the NBODY6++ application use resources in other national and international compute sites (which are not primarily a part of the AstroGrid-D community). The architecture of the information service should acknowledge this by providing a means to cover information about these resources as well.

## UC4 - Dynamo

Dynamo is an application for solving the induction equation with turbulent electromotive force (Alpha tensor) for modelling the turbulent dynamo in planets, stars and galaxies.

The Dynamo scenario is similar to the other simulation use cases wrt. the requirements related to job and data file management operations. Also, the scenario may require certain job progress information to be stored in the information service. In addition, the use case outlines that simulation results shall be used with post-processing. Thus, the information service may store metadata about data provenance. This metadata could include information about the input data, the used program (version), the simulation parameters, etc.

## UC5 - Cactus

Cactus is used by the physicists in the Numerical Relativity department of the AEI to numerically simulate extremely massive bodies, such as neutron stars and black holes.

The Cactus scenario shares most features with the other simulation use cases. In addition, metadata about the nodes' interconnect properties and the available storage or file system resources must be managed by the information service. Cactus itself is currently reporting application-specific metadata to a Cactus user portal which stores such information in an internal database. It includes administrative data (URL of the HTTPD Thorn, access restrictions), periodically updated data on the application's progress (iteration and simulation time, current state) and various simulation parameters (eg. name and version of the Cactus executable, contents of the parameter file, list of output directories and files). In the future, all this information shall be stored in the information service. To facilitate the collaboration of the scientists and to manage a growing number of simulations (eg. parameter studies), it is necessary to associate general grid job metadata (eg. started when/where/how/by whom, resource usage, exit code) with the application-specific metadata as described above. In general, the metadata stored in the information service must be efficiently searchable for users through a portal, for example (a) *"list all simulations of a this Cactus user within the last 2 weeks and show me the parameter files"*, or (b) *"find simulations where this parameter setting had been used and list the corresponding output files of the template given"*.

## UC6 - GADGET

GADGET is a freely available code for cosmological N-body/SPH simulations on massively parallel computers with distributed memory.

The GADGET scenario is similar to the other simulation use cases wrt. the requirements related to job and data file management operations. Because GADGET jobs can use parallel resources, the information service should provide metadata about the node interconnect and the network properties to select the best sites if a job requires multiple resources. The application may be build from the sources at the execution site. Therefore the software and build environment must be described in the information service. For example, the metadata about compute resources could be extended accordingly. Also, the scenario may require certain job progress information to be stored in the information service. In addition, the use case outlines that simulation results shall be used with post-processing. Thus, the information service may store metadata about data provenance. This metadata could include information about the input data, the used program (version), the simulation parameters, etc.

## 2.2 Analyzation of data sets

There are six use cases describing data analyzation runs. The data sets are usually very large (up to Terabytes). Because of limited network bandwidth and traffic volume it is necessary to reduce the amount of data which is transmitted from the data storage (file server or database) to the compute resource where the analysis is performed. While some scenarios explicitly request the movement of the data, others consider the movement of the analyzation code

to the data storage for performing the operations on them (provided that the data storage provides sufficient compute power). The size of the results is usually smaller than the size of the input data. Further improvements may be achieved by collecting information about data provenance. Thus already generated data products may be fetched from a storage rather than generating them again and again.

Similar to the simulation runs the data analyzation use cases require the metadata management service to cope with a wide variety of metadata schema which are not known a-priori (i.e. user-specific description of data sets). Also, the metadata management service must be able to protect the information such that only certain community members may access the data-set-specific information.

### **UC7 - Astrometric matching**

Astrometric Matching for classification of Spectral Energy Distributions (SED) is an essential research technique to discover new astronomical objects like obscured active galactic nuclei (AGNs), brown dwarfs, isolated neutron stars, or planetary nebulae (PNs).

The data about the astronomical objects is usually stored in catalogues. Besides a list of catalogues the information service could store information about the schema of the data objects. For query optimization the scenario uses several static and dynamic information about the computing, storing and networking resources. The static information refers to the maximum capabilities of the resources, for example processor speed, disk capacity or network bandwidth. The dynamic information covers data on the current usage of the resources, for example processor load, available disk space or network traffic. In addition information about the software environment of a resource (in particular for computing resources) is required.

Preservation of the metadata of the input catalogs and software settings that can vary from run to run, such as the stringency or laxity of the matching requirements and how missing data is handled are important to described the output data. Additionally, summary metadata, such as the area of sky covered, the number of matches, and the number of orphaned objects will also be of interest.

### **UC8 - Clusterfinder**

The purpose of clusterfinder is improving the source-identification of X-Ray galaxy clusters by correlating data of X-Ray photon maps (ROSAT All Sky Survey - RASS) and optical galaxy maps (Sloan Digital Sky Survey - SDSS).

The scenario accesses specific catalogues (RASS and SDSS). The job management must be able to verify if the executing machine has network connectivity to these catalogues. Therefore the metadata about compute resources must supply information on the connectivity to the machines storing the catalogues. In addition, a list of catalogues could be used to virtualize the access to the data. Also, this list could provide information on access restrictions of the catalogues. It may be necessary to store logging information or information about logging files. Access to these information is restricted to its owner (user/group based). The main objective of the use case is to speed up the processing of the data by using more compute resources. Because the number of results (files) may be large it is also desired to let a user

verify if some data has already been processed. Therefore, the information service should associate appropriate metadata (input parameters, input data source, program version, etc.) with output results and provide the necessary search capabilities.

## UC9 - Virtual Telescopes

In “Theory VO” (VO=Virtual Observatory !) parlance, a “Virtual Telescope” is a piece of software that mimics an astronomical observatory (telescope/instrument/satellite/spectrograph etc.) which can “observe” the result of a simulation and produce a synthetic observation that can be directly compared to the output of the instrument that is being simulated.

The scenario requires a catalogue of data sets which are the outcome of simulations (the specific simulation could be an attribute to the data). A metadata repository is not developed, but the scenario could benefit by storing the following metadata.

- Existing simulation results.
- Public software capable of creating new simulation results.
- Available virtual instruments.
- Existing virtual observation results.
- Grid resources (computational, data transfer and storage).

Essentially, the scenario is concerned with post-processing the data sets. The scenario mentions visualisation as post-processing, though any post-processing might be possible. The information service must store information on the available compute resources. Because the post-processing programs are compiled from the source code, the IS must maintain meta-data about the software and build environment of the compute resources. The location of the output files (images and log-files) must be stored in the IS as well. To simplify parameter studies the information service should also support data provenance.

The scenario also mentions IVOA standards for modelling, storing and querying meta-data. These could be implemented on top of the IS or the information could be integrated into the IS directly. It should be noted that a clear and complete IVOA standard is not exist currently.

## UC10 - Millenium Query

The scenario Millenium Query is about selecting a subset of a large remote dataset and moving the result to the user.

The scenario does not need descriptions of resources except for the available databases. This is due to the fact that it mainly covers the filtering of the data sets at their storage (relational database system). Because the access to these data sets may be restricted, the ADG IS may store information about users’ permissions. In addition, the IS may store information on the filtered data sets (i.e. results of filtering), which are stored as tables and/or as files. This information could be used to avoid reproducing subsets of the data. The information about

dumb files can be used to fetch them to a local disk via standard Grid tools such as GridFTP. Giving access to many users on the full database, the available disk space (for subsets of the data as table or file) should be monitored and managed via a quota-like mechanism. With larger original data sets and concurrent accesses to them the time needed for filtering requests may increase as well. Thus a kind of batch system or processing time limits are needed to ensure fairness among the users.

### **UC11 - Millenium Post-processing**

The scenario Millenium Post-processing deals with on-demand post-processing of the output data from cosmological simulations, in particular the 25 Terabyte Millenium Simulation run of the Virgo consortium.

The scenario requires the description of compute resources and the installed software/build environment on them. Due to limited network traffic volume between possible computing sites and the data storage, the scenario could benefit from (a) replicating parts of the data to other storages, e.g. near the computing sites, select appropriate data sources through a replica management, and let the Grid job management optimize the placement of jobs, and (b) processed data pieces may be cached at the computing site to minimize data transfers if the same data is used again. Last, to make the results of the data analysis accessible for a wider user community, the information service may store access permissions for the results.

### **UC12 - Geo600**

The gravitational wave detector GEO600 near Hannover and other ground-based devices (eg. LIGO in the U.S., TAMA in Japan, VIRGO in Italy) aim at the direct detection of gravitational waves by means of a laser interferometer. The detectors constantly generate data (eg. LIGO in the order of 1GBytes/day) that needs to be searched for gravitational wave signals (eg. by filtering the raw input data, running various transformations and check for signals).

The scenario requires the description of compute resources and the installed software/build environment on them. As the application reads input data from and writes output data to files in an NFS directory, these shared disk spaces should be described and associated with the compute resources. Currently, the program writes logging information to stdout/err, but does not store these metadata into an information service. In the future, certain information could be logged into the AstroGrid-D IS. Then, it may restrict the access to the owner of the job. The information service may also store information about the replication of input data sets to let the Grid job management optimize the placement of jobs. Last, the results of the data analysis will be published to a wider user community. Thus, the information service may store access permissions for the results.

## **2.3 The Planck Process Coordinator (UC13)**

The Planck Process Coordinator (ProC) is a workflow engine originally developed for running simulation and data analysis workflows (called “pipelines” in astronomical parlance) that occur within the Planck Surveyor project. Because it can be used for both simulation runs and data

analyzation (or even both combined) we did not assign it to one of the previous categories discussed in Sections 2.1 and 2.2. Due to its generic nature, the ProC does not require any feature not already mentioned in the subsections above.

## 2.4 Integration of Robotic Telescopes

The partners maintain or have access to robotic telescopes. These robotic telescopes can be seen as “normal” Grid resource similar to a compute resource. Instead of executing a compute job they perform observations which also produce data. Besides more complex descriptions of jobs, the management of robotic telescopes also differs in that most observations must be performed during certain time windows. Then also the local weather conditions need to be taken into account for Grid-wide job/observation distribution.

**UC14 - STELLA** The AIP maintains a robotic telescope at Tenerife Island. The telescope should be seen as a “normal” Grid resource. Advanced usage scenarios, such as instant reaction to astronomic events or arbitrary long observation of specific objects, become possible with a Grid of world-wide robotic telescopes.

**UC15 - PLANET** The PLANET team (Probing Lensing Anomalies NETwork) uses five telescopes in the southern hemisphere to monitor microlensing events that had been alerted by some other team (OGLE) in order to detect anomalies in the lightcurve, which could be due to the lens being a double star or having a planetary companion.

## 3 Requirements on the Information Service

### 3.1 Grid and Generic Information Service Requirements

The following requirements are derived from the need of the information service and other grid middleware components. Additionally, more specific requirements related to the use cases are presented in section 3.2. The numbering of grid requirements is sequential with the prefix "GIS". Some of these requirements may seem obvious after the previous sections but are included as requirements for completeness.

- GIS1** An information service must have functionality to store metadata.
- GIS2** An instance of the information service must provide a remotely accessible interface including methods for management of the stored metadata, addition of new metadata and access to stored metadata.
- GIS3** The remotely accessible interface must support X.509 authentication and authorization of clients. This requirement is based on the grid vision off Single-Sign-On.
- GIS4** Due to the volatile behavior of grid resources and therefore also the metadata related to these resources it should be possible to specify a life-time for metadata.
- GIS5** The information service is vital for the discovery of grid related metadata. Therefore, service availability and data durability are important for increased overall grid reliability.
- GIS6** Collaboration between individuals and institutes part of AstroGrid-D, but also with external partner, is a key to increased scientific quality and progress. The implications on the information service is that metadata should be easy to extract and be provided in a portable format.
- GIS7** The external service interface should build on well-established techniques used within the grid or web service community such as SOAP [21] or REST principles [18].
- GIS8** It is expected that the AstroGrid-D community will grow over time, both with regards to user, resources and services, why it is important that also the information service can scale to include more information and handle increasing usage.

### 3.2 User Requirements

The following requirements are specific to the Information Service within in AstroGrid-D. They are specific to the use cases summarized in section 2. The numbering of the user requirements is sequential with the prefix "UIS". Note that this list is in no way exhaustive, but a subset of

the most significant requirements. Also, it should not be seen as the final list, but as a current snapshot derived from the available use cases.

- UIS1** Exact key query, a search on an exact key word that returns all values related to the given key, example "return all cars manufactured in 1967".
- UIS2** Range query, a search over an interval that return a set of values, for example " return all cars manufactured between 1950 and 1955".
- UIS3** Discovery of metadata available in application specific files, such as FITS and HDF5.
- UIS4** Metadata vocabularies and schemas should be easy to extend. Several users have not defined their metadata schema yet.
- UIS5** The Information Service should be extensible to be able to store metadata entries from currently unknown vocabularies.
- UIS6** Entries of metadata should be available to non-AstroGrid-D users under the assumption that this is allowed by the owner of the metadata.
- UIS7** Metadata access should be restricted to authorized users and groups.
- UIS8** It should be possible to apply fine-grained access restrictions to the metadata.
- UIS9** It should be possible to add metadata related to special hardware attached to a host.
- UIS10** Support for discovery of non-standard resources such as robotic telescopes.

## 4 Related Work

This chapter cover some of the more notable generic metadata storages and information services used in different grid architectures.

### 4.1 Metadata storage

A grid metadata service is responsible for managing a metadata storage and enabling discovery of metadata in the storage. Standard DBMS systems such as IBM DB2 or Oracle data bases are often not fulfilling the needs of a grid metadata service for a number of reasons.

#### **ARDA Metadata Grid Application**

The ARDA Metadata Grid Application (AMGA) [28] is developed at CERN and is part of the gLite grid middleware as of version 1.5. The main design goals are to expose a simple interface for users to relieve them from the internals of the service, to support flexible and dynamic schemes, be scalable enough to deal with million of entries and to be able to structure the metadata in a hierarchy. Their data model is rather simple; an entry is the name of a data item or resource, an attribute is a (key, value)-tuple and a schema is a logical collection of attributes. The service interface allows for adding and removing entries, attributes and schemas. They also have a fine-grained security system with the possibility to define user, groups and ACLs. Metadata querying is done through an AMGA specific query language which have some similarities with SQL, providing both multi-attribute and range queries. The interfaces are exported in both SOAP and an AMGA specific ASCII based client/server protocol. Current work on AMGA is focused on replication of metadata to increase scalability and reliability.

#### **Metadata Catalog Service**

The Metadata Catalog Service (MCS) [16] was developed by support from several grid projects and is stand-alone but can be used in conjunction with the Globus Toolkit [2]. MCS was designed to store and share metadata, allow for easy changes of metadata schemes and to support metadata about large-scale data sets. The MCS data management model is based on three main elements: data items, collections and views. A data item is the basic object within the model and may for example represent a file or an image. The collection concept is a mechanism to name a set of data items. A collection can also group other collections. Views allow a user of the MCS to group collections and data items indepent of already existing collections. Querying in MCS is done by specifying a set of conditions and the return type (data item, view or collection).

## Summary

AstroGrid-D requires a metadata service where arbitrary metadata can be stored. The minimum query requirements in AstroGrid-D is multi-attribute queries and range queries. MCS cannot be used within AstroGrid-D since it has very limited range query capabilities. AMGA, on the other hand, seems promising and more feature-rich than the current MCS implementation (version 3.1). The main reason against using AMGA in AstroGrid-D is collaboration. Metadata stored in and extracted from AMGA is in an AMGA specific format. This may not make any difference if the service is only used within a project, but for AstroGrid-D where users from other environments will access the metadata it is better to use a standard format. Another issue is the use of a relational model, which does not fit for metadata. Metadata entries may not always contain values for all attributes in a relational table since they are often provided on a best-effort basis. The consequence is that tables will contain empty cells, which is inefficient from both storage and access perspectives.

## 4.2 Grid information services

Resources and grid specific services generate metadata about their current status and availability. A grid information service is responsible for storing this information and for making it possible for both grid components and users to access the stored information. The information services presented here are the most relevant for the AstroGrid-D project.

### Monitoring and Discovery Service

The Monitoring and Discovery Service (MDS) of Globus Toolkit 4 (GT4) [2] is focused on monitoring and discovery of grid resources. MDS is mostly used in the context of resource selection. It aids the user or agent to identify the hosts on which to run a specific application. The MDS architecture basically consists of three different components. Information providers are resources such as hosts, services or tools. MDS4 services providing an Index Service and a Trigger Service and WebMDS which is a web-based interface for the Index Service. For collecting data from the information providers, the MDS4 services are built on a more general framework, the Aggregator Framework. An MDS4 Service acts as aggregator service to collect data from the providers which are accessed through aggregator sources. An Index Service can also act as an information provider, allowing a hierarchical structuring of Index Services. This might be useful in situations where a site has several nodes where some of them are not public to the Internet. Stored data always have a life-time which makes it possible to deploy a self-cleaning mechanism to get rid of old information. Information may be collected from the providers in different ways: by accessing them through WSRF [19] services or by running an external, administrator-supplied executable. The WSRF services provide either a periodical polling of information or a subscription/notification mechanism, allowing information to be updated when information changes. GT4 also provides implemented aggregator sources, like the Grid Resource Allocation and Management service (WS GRAM).

## R-GMA

The Relational Grid Monitoring Architecture (R-GMA) [31] is based on the Grid Monitoring Architecture (GMA) [30]. R-GMA's implementation is based on the relational information model (tables, attributes and rows) and supports a subset of SQL for querying. This allows for more powerful queries to an information and monitoring service, which is the main goal of R-GMA. The GMA model consist of three parts: consumers which requests information, producers who provide information and a registry which acts an intermediate between consumers and producers. R-GMA plays an important role within the EGEE middleware architecture [1].

## GridICE

The GridICE system [4] is developed mainly for monitoring resources within a grid environment. System requirements were derived from demands of three different user categories: Grid operators, virtual organization managers and site administrators. From these requirements, ranging from presentation issues to requirements such as resource metadata history and detection of fault situations, a generic model based on four components was derived. The four components include: sensors that do the actual measurements, producers which makes sensor data available through an interface, republishers that reorganize producer data and makes it available through a different interface and consumers that access the information via republishers. The architecture has two levels; one site-level that represents an administrative site or domain and an inter-site level representing the entire grid. Their architecture also accounts for a presentation layer through a common data aggregation and abstraction layer for extraction and translation of metadata into different formats. Flow of information between components can both be pull/pushed-based and use the publish/subscribe model. GridICE is also used within the EGEE project.

## Summary

The AstroGrid-D information service has very mixed requirements (collaboration, heterogeneous environment, dynamic and static metadata, ...) and none of the presented systems can cover all of them. One alternative is to use two different systems, one as a metadata service and one for monitoring and discovery of grid resources. This solution has the disadvantages of forcing developers to use two different interfaces and also the existance of two different software systems that need maintainance and support. Even though the AstroGrid-D metadata have different characteristics (life-time, hierarchies, ...) and different ways of production and access (resource information is collected by a resource monitoring system and scientific annotations are typically made by a researcher or domain-specific application), a layered design with mechanisms to support these different characteristics is a better approach then a fragmented system. The AstroGrid-D information service takes inspiration from general metadata services and grid monitoring services and enables a unified service for metadata storage and querying. This is accomplished by using a powerful intermediate metadata representation combined with a simple query language.

## 5 AstroGrid-D metadata

Metadata is information about information. From a more pragmatic point of view, metadata can be "tagged" to other information with the purpose of giving additional value to the information. EXIF is a good example of metadata with the purpose of adding more information to images. Image metadata can for example include the date when the image was created, image structure format and even the GPS location of where an image was created. Metadata within AstroGrid-D is focused on lower level metadata such as file format descriptions and grid resource information, but also includes higher level metadata like application-specific metadata and scientific metadata. This chapter gives an introduction to what type of metadata exist within AstroGrid-D. It also gives an introduction to RDF, the metadata information model for AstroGrid-D, how to define metadata schemas using RDF Schema and how to query RDF metadata through SPARQL.

### 5.1 Metadata classes

In AstroGrid-D, the main purpose of metadata is to enable discovery information related to scientific information, application information, grid component information and resource information. This classification of metadata is user centric, meaning that the focus is on the questions by whom the metadata is used and how the characteristics between these different classes differs.

#### Scientific

Scientific metadata describes datasets related to scientific data. This metadata should provide sufficient information for the scientist, in this case a domain-expert, to easily locate the relevant datasets. Scientific metadata is typically write once, read many (WORM). An important aspect of scientific metadata in AstroGrid-D is the security concerns related to access of metadata and datasets.

#### Application

Application metadata is the metadata describing individual applications run history with input parameters and result location. This metadata is mainly there to allow the researchers to keep track of how they have used their application to avoid re-runs of compute and storage extensive jobs. It can also be used for easy retrieval of already generated data. The characteristics of this metadata is typically write once and read many.

## Grid

Grid metadata reflects the current state of the grid services relevant for the AstroGrid-D middleware. The main purpose of this metadata is outlined by the responsibilities of the individual grid services. Due to the large variation of grid services it is difficult to label this metadata as either write- or read-intensive. Although, it is possible to state that this metadata is typically temporary in the system since it reflects the current state. The typical life-time is difficult to assess since, for example, a logging service can require to store a history trace of the state. An example of grid metadata are the current grid jobs stored in the Information Service by the job management. This metadata is updated often, but may also be consumed often by clients such as the user portal.

## Resources

Resource information describes the different resources present condition. It is mainly used by other grid services for taking decisions about job and data placement, but also for supervision and status control of the grid. The metadata characteristics are dependent on the resource type. Some parts of the metadata for each resource is updated dynamically with frequent changes and some parts are static.

## 5.2 Metadata overview

### 5.2.1 Resource specific metadata

AstroGrid-D includes three different type of resources: computational, storage and observational. Computational resources range from High performance computers and clusters to individual workstations. Storage resources include file storage on the different compute resources. An observational resource is a type of sensor, an observational resource within AstroGrid-D is typically a robotic telescope. The resource specific metadata is currently more well-defined than the metadata from other metadata classes.

**Computational resources** are everything from High Performance Computers (HPC), Clusters and individual workstations. (A computational resource can also be special hardware attached to a compute site.) The common schema include both static metadata such as processor type, processor speed, number of processors, total memory and dynamic metadata such as job queue, current processing usage and current memory usage.

**Storage resources** include static metadata such as total storage available, available access protocols and file policies. The dynamic metadata can for example be the currently available free storage.

**Network monitoring** should be performed on a grid-wide basis. This includes metadata about link connectivity, round-trip time and throughput.

**Observational resources** are special for AstroGrid-D and are characterized by both static and dynamic metadata. Static metadata can for example be restrictions in observation area, while dynamic metadata can be current weather information.

The GLUE schema [3] is a general information model containing the most common attributes used for describing computational and storage resources. It was developed with the goal of creating better interoperability between grid resource and monitoring systems. Another advantage is that resources that are part of more than one grid can provide its information in a common format to different Information Services. Since the GLUE schema is quite large, a short description of its capabilities are given here.

A Computing Element (CE) from the GLUE schema describes a job queue managed by the Local Resource Management System (LRMS) at a computational resource. A CE entry contains information that is important for resource selection, in respect to resource matching and meta-scheduling, local CE policies and the current state of the CE. The attributes interesting for resource matching is for example software such as operating system and available libraries, processor type, speed and architecture, and main memory size. Meta-scheduling is a term for grid level scheduling. Relevant information for meta-scheduling included in a CE description is also similar to the current state of the CE. This is mainly free job slots, waiting and currently executing jobs, CE estimated and worst response time. Other information that is part of the GLUE schema for a CE includes structure of a cluster (number of CPUs), data directories for temporary data, LRMS information and identification data.

A Storage Element (SE) is called a storage resource within AstroGrid-D. SEs can be everything from simple disk servers to Storage Area Networks (SAN) and are often attached to a CE. What characterize an SE is mainly location, total and available storage space, storage policies and access and control protocols. For example, the access protocol information includes supported data transport protocols (FTP, GridFTP, HTTP) and what capabilities they support (partial file transfer, multi-streaming, ...). SE metadata is relevant for both the file management system and the job management. The GLUE schema also includes a relation between SEs and CEs. This enables the job management to choose a CE with one or more corresponding SEs that complies with the storage requirements of a job.

In addition to the GLUE schema AstroGrid-D need resource information that describes existing special hardware, network information and the observational resources. This information is used to select the correct resource and is vital to support the different applications foreseen to execute on AstroGrid-D resources. Any additional information providers derived from the observational resources, network information or metadata regarding special hardware should either design an individual schema or extend the GLUE schema where necessary. Network information is typically the current usage and total capacity of links between different hosts at a computational resource or between AstroGrid-D computational resources. Special hardware can for example be GRAPE or FPGA boards and are typically associated with a CE.

## 5.2.2 Integration of Robotic Telescopes

Observational resources within AstroGrid-D are currently represented by robotic telescopes. They have a predefined schema for discovery called "phase-0", which provides the requester with a description of the services provided by a telescope. This metadata is presented in an XML format called RTML [23].

Robotic telescopes can be viewed as resources with very special hardware, thus resource discovery should allow the determination if a 'job' (here an astrophysical observation) can run on the robotic telescope in question (see phase 0). A robotic telescope will always expect a relatively

```
<?xml version="1.0" encoding="ASCII"?>
<RTML version="3.1a" mode="phase0"
uid="org.telescopenetworks://query-2006-01-23T01:23:45">
<History>
<Entry timeStamp="2006-01-23T01:23:45" purpose="phase0">
<Agent name="estar.astro.ex.ac.uk://8080" /></Entry>
</History>
</RTML>
```

Figure 5.1: An RTML-resource discovery query.

small set of input data, where the details of processing are hidden from the user. Thus, the input data must be both, generic enough to be executed on different robotic telescopes, but nevertheless specific enough to allow a translation to the telescope-specific execution queue.

Most observations will last rather long, and considering the comparably high possibility of execution failure, most telescopes will already facilitate some sort of monitoring mechanism. On-line steering will at least include execution abortion, but some mechanism to change queuing priorities might be present as well. Steering will only be allowed by the owner of a target, thus protecting the metadata information is a very sensible item.

### Resource discovery

Resource discovery for robotic telescopes means to detect those telescopes providing the hardware needed for a special observation request. Efforts are underway to standardize hardware description using XML documents. In particular, the heterogeneous telescopes network (HTN) already put together an RTML (Remote Telescope Meta Language)-based schema on how to describe resources. This description is mainly static, it changes only when the hardware of the telescope is changed. An example of a resource query and a valid answer is given in Fig. 5.1 and Fig. 5.2, respectively.

Range queries must work on enumeration types, i.e. in Fig. 5.2 a range-request can query for the availability of filter type of *U*, which would cover a range of *Bessel-U* to *Johnson-U*.

Weather data is currently not supported in HTN and thus has no representation in RTML. The main reason behind this it that weather data is considered to be too volatile and unpredictable to include in resource discovery.

Access to telescope resources in the HTN for the time being is by registry entries giving the resource node with its port, the protocol plus additional data required by the specified protocol, like an URN on SOAP: `stella.aip.de:7070 soap urn:/rtml_node handle_rtml`.

### Domain specific and Grid metadata

Scientific, application and grid metadata is not as well-defined as the resource metadata. Grid services make metadata available that allow grid users to follow the actual status of, for example, job submissions and file transfers. Metadata about the services themselves, such as name, location and service description, should also be available for discovery. More specific

```

<?xml version="1.0" encoding="ASCII"?>
<RMTL version="3.1a" mode="resource"
uid="org.telescopenetworks://query-2006-01-23T01:23:45">
<History>
<Entry timeStamp="2006-01-23T01:23:45" purpose="phase0">
<Agent name="estar.astro.ex.ac.uk:8080" /></Entry>
<Entry timeStamp="2006-01-23T01:23:50" purpose="resource">
<Agent name="MyTelescope.org" /></Entry>
</History>
<Project>
<AckText>These observations were made using MyTelescope.org facilities.</AckText>
<Contact>
<Institution name="www.aip.de" type="educational" />
<Communication>
<AddressLine>An der Sternwarte 16, D-14482 Potsdam</AddressLine>
<CountryCode>GER</CountryCode>
<Email>stella@aip.de</Email>
</Communication>
</Contact>
</Project>
<Telescope name="STELLA-I">
<Aperture type="geometric" units="meters">1.2</Aperture>
<SpectralRegion>optical</SpectralRegion>
<Camera name="WIFSIP">
<Description>WiFSIP</Description>
<Detector>
<NumColumns>4096</NumColumns>
<NumRows>4096</NumRows>
</Detector>
<PlateScale units="arcseconds_per_pixel">0.32</PlateScale>
<FilterWheel>
<Filter type="Johnson-U" name="U"/>
<Filter type="Johnson-B" name="B"/>
<Filter type="Johnson-V" name="V"/>
<Filter type="Cousins-R" name="R"/>
<Filter type="Cousins-I" name="I"/>
<Filter type="Sloan-u" name="uprime"/>
<Filter type="Sloan-g" name="gprime"/>
<Filter type="Sloan-r" name="rprime"/>
<Filter type="Sloan-i" name="iprime"/>
<Filter type="Sloan-z" name="zprime"/>
<Filter type="Stromgren-u" name="u"/>
<Filter type="Stromgren-v" name="v"/>
<Filter type="Stromgren-b" name="b"/>
<Filter type="Stromgren-y" name="y"/>
<Filter type="Halpalpha" name="Han"/>
</FilterWheel>
</Camera>
<FocalLength units="meters">9.600</FocalLength>
<FocalRatio>f/8</FocalRatio>
<Location name="Izana Observatory, Teneriffa, Spain">
<Latitude units="degrees">28.3</Latitude>
<EastLongitude units="degrees">-16.509722</EastLongitude>
<Height units="meters">2480</Height>
</Location>
</Telescope>

```

Figure 5.2: An example reply to a phase-0 request, as defined in the HTN standard.

service metadata such as access statistics can be stored at the service. Schemas for grid service information is not clear at the time of this writing. The same applies to application and scientific metadata which is covered in chapter 2.

### 5.3 Representation of Metadata

Metadata in the Information Service is represented using the Resource Description Framework (RDF) model developed by W3C [25, 5, 22]. The motivation for choosing RDF is based on several important factors. First, RDF was developed to annotate information, from concrete objects such as files to more abstract concepts and ideas, but also to be easily exchangeable and interpretable by machines. Second, the metadata requirements from the initial AstroGrid-D user base is highly heterogeneous. Metadata range from annotations in file formats such as HDF5, FRAME and FITS to more abstract annotation requirements such as simulation history. Furthermore, several of the users have not defined their metadata format yet and leave this as an open issue for future development. Both these conditions lead to the conclusion that the metadata representation format has to be very flexible. Finally, the AstroGrid-D middleware provides a platform for collaboration and a non-proprietary solution for metadata representation is a viable choice for increased interoperability.

The atom in the RDF model is a (subject, predicate, object)-tuple usually referred to as triple or statement. An RDF triple can be derived from normal human language statements, for example "AstroGrid-D has a member institute called ZIB". In this sentence "AstroGrid-D" is the subject, "member institute" is the predicate and "ZIB" is the object. RDF allows multiple statements about a single subject, for example "AstroGrid-D has a member institute called AIP" and "AstroGrid-D has a website at <http://www.gac-grid.de>". It is also possible to make statements about objects, "ZIB has an address with value Takustrasse 7, D-14195 Berlin-Dahlem". The possibility to make statements about objects is one of RDF's strengths and provides an easy way of extending metadata entries with further meaning.

Listing 5.1: RDF example

```
@prefix agde_terms: <http://www.gac-grid.de/terms/#> .
@prefix dgrid: <http://www.d-grid.de/projects/> .

dgrid:AstroGrid-D agde_terms:memberInstitute <http://www.zib.de> .
dgrid:AstroGrid-D agde_terms:memberInstitute "AIP" .
<http://www.zib.de> agde_terms:address "Takustrasse 7, D-14195 Berlin-Dahlem" .
```

Figure 5.3 shows a graph representation of the RDF statements in listing 5.1. This listing is a more formal representation using triples notation of the example statements made in the previous paragraph. The listing contain two definitions of prefixes and a set of statements using the prefixes. Both representations use URI references for identification of subjects, predicates and objects. A URI is a pointer to an arbitrary resource, for example a grid site, a job or a specific file. URI:s are supposed to be unique and does not necessarily reference resources reachable via a network.

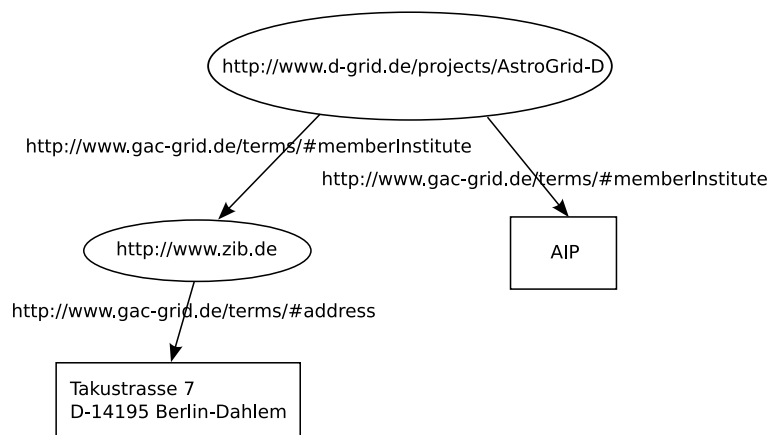


Figure 5.3: An RDF example graph.

## 5.4 Discovery

Discovery is the ability of locating data or metadata through either search or browsing. Search or querying is provided directly by the Information Service interface (section 7) and browsing can be implemented by using the query interfaces provided by the Information Service. This section focuses on what type of search is needed within AstroGrid-D and gives a basic introduction to the SPARQL query language that is used for query requests to the Information Service.

Search of metadata in AstroGrid-D can be motivated by several examples. The metadata that is stored within the Information Service should be rather clear by now from the previous sections. What might not be clear is how a data set created on December 24th between 18:00 UTC and 19:00 UTC is found or how a compute resource with 12 free processors and 2GB internal memory is found. More specifically, how does the Information Service search operation function?

The Information Service should have support for both multi-attribute and range queries. A multi-attribute query is given a set of (attribute, value)-pairs. The result is a set of entries which complies to the given query. The example above with the attributes free processor = 12 and internal memory = 2GB could for example return two entries: (job queue = A, host = hpc.example.com) and (job queue = B, host = cluster.example.org). A range query is given attributes with a minimum and maximum value, i.e. a range or interval. The previous example could be translated into time min = December 24th 18:00 and time max = December 24th 19:00 and, for example, return a set with one entry: (data set ID = 12345ABC).

The Information Service provides support for the query language SPARQL (pronounced "sparkle") [27]. This language is specifically designed to match RDF graph patterns and was developed by the W3C. Implementations of SPARQL exist for several RDF stores. Multi-attribute and range queries are a subset of the functionality provided by SPARQL. Even though many of the features provided by SPARQL are not part of any AstroGrid-D use case it is anticipated that parts of the feature set might be needed at a later stage. An example of such a feature is the possibility of extensible value testing. This feature provides a way to define comparison

methods for arbitrary datatypes which then can be used for result filtering. Another reason for choosing SPARQL is that it is backed by the W3C and is therefore becoming the de-facto standard query language for RDF stores.

SPARQL is designed to match RDF graph patterns. An RDF graph is a set of RDF triples and a graph pattern is simply a triple pattern (subject, predicate, object). It is possible to replace any of the terms in the RDF triple with a variable. A variable can then match any value in the RDF graph that corresponds to the variable position (i.e. subject, predicate, object). There can also be more than one variable in an expression making it possible to match more terms at the same time. A simple example query, using the data presented in listing 5.1, is in listing 5.2. The query itself says: "List all member institutes of the D-Grid project AstroGrid-D.". Two prefixes are defined first for convenience when formulating the query. In the query itself a variable is used at the object position. This means that all triples that have the fixed subject "dgrid: AstroGrid-D" and the fixed predicate "agde\_terms: memberInstitute" should be part of the result.

Listing 5.2: Simple SPARQL example.

```
PREFIX agde_terms: <http://www.gac-grid.de/terms/#>
PREFIX dgrid: <http://www.d-grid.de/projects/>
SELECT ?institute
WHERE { dgrid: AstroGrid-D agde_terms: memberInstitute ?institute }
```

<b>institute</b>
AIP
<http://www.zib.de/>

Table 5.1: Simple SPARQL query result

More advanced queries can have more variables, use conjunction to provide more patterns to be matched, use optional result patterns, use filtering on values and take input from multiple sources. SPARQL supports four different type of queries. The SELECT query which returns a set of result entries with columns corresponding to a defined set of variables. In listing 5.2 this set contained the variable "?institute" and the resulting entries are shown in table 5.1. The SELECT query is probably the query used in most cases. Another query type is CONSTRUCT, which given a graph template can create new a RDF graph. This can for example be useful when making transformations between different standard descriptions. DESCRIBE is another query type. DESCRIBE returns a new RDF graph with information about resources defined in the query. A DESCRIBE query asking about "dgrid: AstroGrid-D", from the previous example, would return RDF statements regarding all of the member institutes. The last query type is ASK. ASK returns a true or false depending on if the given pattern matches or not. Both SELECT and ASK can return values through a W3C specified XML result format [7].

## 5.5 Metadata schemes

A schema or vocabulary is a set of defined terms to describe a specific resource. Using common terms for describing things or resources within a community is very important for communication. Biologists, for example, have an enormous schema/taxonomy and common vocabulary

for describing species. Different classes and properties are distinguished through terms in a vocabulary. Another example, tables are a sub-class of furniture and have properties such as height, width and colour. The RDF specification includes a common vocabulary called RDF Schema (RDFS) [10], which is used to describe classes and properties. RDFS does not describe specific classes and properties such as table and colour, instead it includes a vocabulary to describe classes and properties themselves. Another interesting "feature" of RDFS is that from an RDFS description it is possible to derive which classes and properties belong together. Basically, RDFS provides additional information to RDF statements that indicates which class they belong to and how their properties should be used. RDFS is the type schema for RDF.

RDFS provide terms for describing classes, class hierarchies and properties. Classes are simply types or categories of things, similar to object-oriented programming languages. A property is a characteristic of a specific class or category. A typical property, taken from the table example above, is colour. Figure 5.4 and listing 5.3 is an example schema defining two type of tables, a desk and a dining table, and the property number of seats. The `rdf` and `rdfs` prefixes defined in the first rows are referencing existing vocabularies for both RDF and RDFS. As can be seen in the example, RDFS include terms for creating class hierarchies with classes and inheritance (sub-classes). It also includes terms for defining ranges and domains of properties. A range defines what values a property can have (number of seats is typically an integer) and domain indicates that a property applies to a certain class, in this case, number of seats refers to seats around a table.

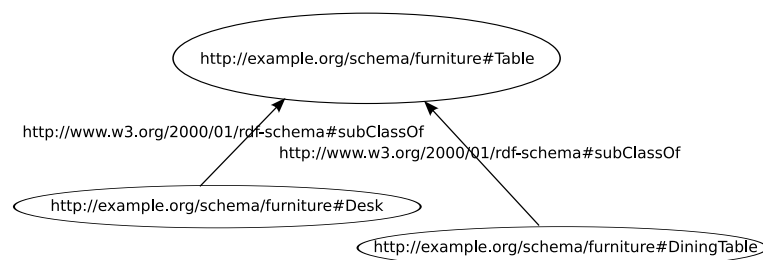


Figure 5.4: An RDFS example graph.

Listing 5.3: Simple RDFS.

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX example: <http://example.org/schema/furniture#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

example:Table          rdf:type          rdfs:Class .
example:Desk           rdf:type          rdfs:Class .
example:DiningTable    rdf:type          rdfs:Class .

example:Desk           rdfs:subClassOf  example:Table .
example:DiningTable    rdfs:subClassOf  example:Table .

example:noOfseats      rdf:type          rdf:Property .
example:noOfseats      rdfs:range          xsd:integer .
example:noOfseats      rdfs:domain          example:Table .

```

An RDFS schema is not prescriptive, it is descriptive. The difference is that a prescriptive schema poses constraints on the instances of a schema. For example, if an RDFS schema would be prescriptive, then an instance of the schema in listing 5.3 must contain a value for the property `example:noOfSeats` to be created. Now, since RDFS is descriptive this is not necessary. An RDFS schema is only providing extra descriptions for RDF resources. An application can interpret these extra descriptions in any way it finds suitable. For example, within scientific schemes it is extremely important to ensure that values are of correct type (meters, kilograms, ...). Therefore, an application using RDF metadata from scientific schemes should interpret schema instances in a prescriptive way, i.e. ensure the schema constraints. Using RDFS for describing vocabularies may not always be sufficient since it does not include support for cardinality constraints on properties (there can be exactly one biological father) or specifying that a property is a unique identifier for an instance. These two and other more complex capabilities for schema description are provided by ontology languages, for example OWL [29]. RDFS and OWL schemes are used by an inference engine or reasoner to detect statements that are breaking schema rules. It can also be used to add results to a query by using deduction. The AstroGrid-D Information Service storage will not use OWL, and RDFS will be used only partially (checking standard types such as integer and double) since it can severely impact on the storage performance. This feature can easily be added at a later stage if necessary.

A growing set of de-facto vocabularies from different areas already exist. One of the more well-used is Dublin Core [9] which provide terms for describing document-like objects such as scientific articles, books and so on. Example-terms from Dublin Core include creator, title and format. The FOAF or friends of a friend vocabulary [11] is also very popular and include terms for describing persons, contact information, connection to other persons etc. . More interesting for the astrophysics community is the IVOA Unified Content Descriptors (UCD) vocabulary [17]. UCD is a controlled vocabulary that contains terms for describing astronomical data quantities such as temperature, instrument parameters and positions. The current version of the UCD is not described using RDF and/or ontologies, but this is planned in a future version.

It is not necessary to create a schema to use the AstroGrid-D information service. RDF statements can be added and removed freely without support of a schema. However, it is recommended to design a schema and use a common vocabulary to describe what is inserted. In that way others can easier benefit from the information part of the Information Service storage. Special care must be taken when designing a schema/ontology using RDFS and extensions like OWL. It is recommended to look at the RDF primer for a pragmatic introduction and links to other resources [26]. Already existing schemes and vocabularies, as those presented earlier, should be used whenever possible.

## 6 Architecture

The goal of this chapter is to give an overview of the overall AstroGrid-D architecture and to present the architecture and design of an Information Service used within AstroGrid-D. An Information Service is mainly responsible for management and discovery of metadata. As noted in the previous chapters, the different applications and services of AstroGrid-D have widely differing demands on an Information Service. Some examples are handling metadata which have varied characteristics for reading/writing and different life-time, extendability/flexibility of schemas and the interoperability for exchange of metadata. As a consequence of this, a generic metadata description framework, RDF, developed by the W3C, was chosen for metadata representation. Discovery of metadata represented by RDF model is done by using SPARQL, pronounced "sparkle", a query language specific for RDF. The architectural design make use of open standards to provide a platform that is easy to extend and interoperable with other designs and grid platforms with minor or no modifications.

### 6.1 Architecture Overview

The AstroGrid-D middleware adopts the concept of Service Oriented Architecture (SOA) in which one or more autonomous services are deployed to bring the expected functionality to the end-user. Each service implements a set of methods defined by an interface contract through which other services and applications can communicate with the service. AstroGrid-D consists of a set of core services that exports a set of interfaces which are combined into the AstroGrid-D middleware. These core services are responsible for basic grid mechanisms such as meta scheduling for jobs, data management, stream handling, job monitoring/steering and security management. The combined AstroGrid-D middleware can easily be extended by adding additional services.

A service exports a public interface consisting of methods for the service information and service logic. The latter exported methods represent the functionality provided by a service, for example transformation of information or modification of the service state. Service information methods define ways of how to retrieve state information from the service. The information flow can be either pull-based, initiated by the requester, and/or event-based which is initiated by the service after demands made by a requester. The latter type of information flow is also often referred to as publish/subscribe or notification-based information retrieval. Allowing different type of information flows, make it possible to use resources more economically, and it can also enable efficient methods for information aggregation and service monitoring. Each service also provides mechanisms for authentication and authorization of users. These mechanisms are not part of the public interfaces. Instead, security information is passed within each service request or through pre-initiated sessions. Furthermore, an AstroGrid-D service can be either stateful, by using the Web Services Resource Framework [19], or stateless depending on the requirements. SOAP [21] should be used for messaging and WSDL [14] should be used for

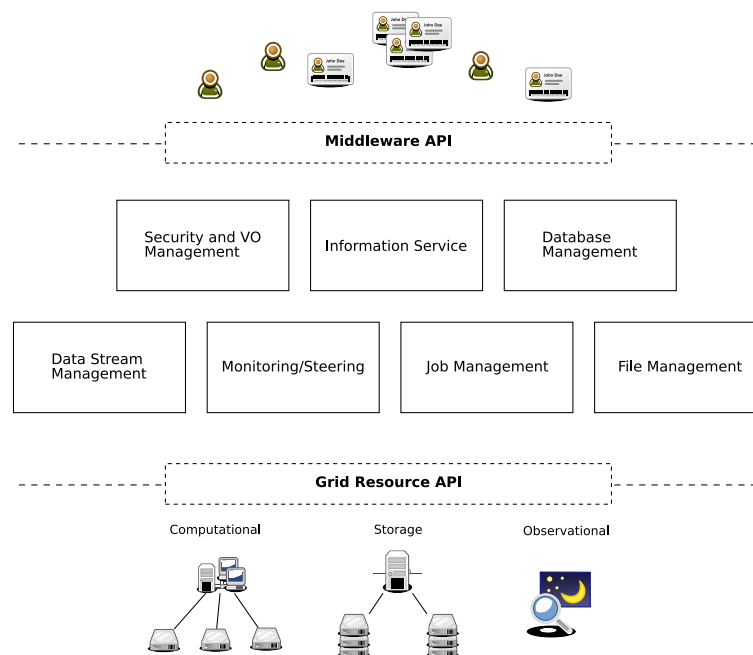


Figure 6.1: A coarse overview of the AstroGrid-D middleware.

interface definitions. An alternative approach is to use REST [18] principles for service design. Both SOAP and REST interfaces can be provided.

All services (job-, file-, data-, stream-management, monitoring/steering and security) use the Information Service to publish information and to enable discovery of their published information (i.e. a job or the physical location of a file). This is accomplished by letting each service export an interface defining methods for information publishing and discovery. For example, the job management could have an information interface exporting methods to add a new job to the information service and to discover these jobs. The Information Service itself is dependent on the Security and VO management service for user authentication and authorization.

Figure 6.1 shows an overview of the most significant services within the AstroGrid-D architecture. The Middleware API is used by a set of users both with and without grid certificates. Users without certificates are restricted to the services and information which is public within the grid. This is to encourage collaboration and to make results easier to publish publicly for the certified AstroGrid-D users. At the bottom of the picture are the resources which range from different computational resources, storage resources to observational resources such as robotic telescopes. They interface to the grid services via a resource API containing methods for resource information dissemination, job handling and data handling. Each service and a brief summary of their main responsibilities is presented below. Note that the ordering in figure 6.1 do not imply any hierarchy and flow of information between the services.

**Information Service** - Supports the grid services, applications and application users with functionality for storage and discovery of information.

**Job Management** - Handles job submission and scheduling of jobs based on current information about the grid resources.

**File Management** - Is responsible for file transfers and replica management.

**Data Stream Management** - Handles efficient distributed data stream processing.

**Database Management** - Responsible for database access from the grid.

**Monitoring/Steering** - Provides mechanisms for monitoring of jobs and grid resources and steering of currently active grid jobs.

**Security and VO management** - Manages users and Virtual Organizations within AstroGrid-D and is also responsible for user authentication and authorization.

## 6.2 The AstroGrid-D Information Service

### 6.2.1 Design objectives

The objectives presented here are derived from the functional requirements presented in section 3, but they also consider the Internet-based environment where the Information Service will be deployed, the expansion of AstroGrid-D, users from different organizations and standard requirements on databases. Note that the following objectives are long term and can be seen as a final goal for the Information Service. This document as well as the implementation is incremental, which is why functional description of some objectives is not presented. The requirements presented earlier in section 3.1 and 3.2 are referenced in paranthesis with their corresponding identifier to leverage their importance in the derived design.

**Service interface.** The external interface must support methods for management and discovery of metadata. It should be implemented using WSDL and SOAP to be compliant with the rest of AstroGrid-D. (GIS1, GIS2, GIS3, GIS7)

**Metadata characteristics.** The metadata storage should provide mechanisms to handle different characteristics of metadata or allow for external services to handle these characteristics. This can include dynamic metadata with frequent updates or static long-lived metadata. (GIS4, UIS9, UIS10)

**Storage distribution.** By deploying a distributed storage increased reliability and scalability can be achieved. Distribution should include at least the metadata to avoid a single point of failure, but this can be extended to also support distributed query-logic. (GIS5)

**Schema extensibility.** It must be easy to extend and change metadata schemes and vocabularies. (UIS3-5, UIS9)

**Collaboration.** Non-AstroGrid-D and AstroGrid-D users must be able to access and easily exchange metadata stored in the Information Service. (GIS7, UIS6)

**Storage reliability.** Individual storages must support backups and durability of stored data. Replication of data should be used to increase reliability. (GIS5)

**Scalability.** Adding new nodes to a distributed Information Service should lead to increased storage and/or query load-sharing with a negligible query-performance decrease or network bandwidth increase. (GIS8)

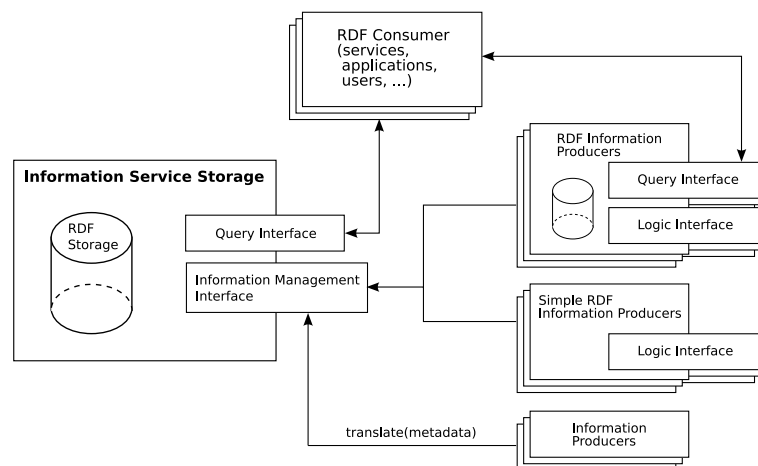


Figure 6.2: The Information Service architecture.

**Security.** There should be support for ACLs on a fine-grained level to comply with the user requirements. There must exist a mechanism to distinguish between public and non-public metadata. User authentication and authorization must be done via certificates (typically X.509). (UIS6-8, GIS3)

**Easy deployment.** No or very small amount of configuration should be necessary to add new nodes to a running Information Service. (GIS8)

## 6.2.2 Architecture components

Figure 6.2 shows the different components and their interactions for the AstroGrid-D information service. The design is influenced by the generic GMA [30] model, including producers/consumers and intermediates which have both functions. This section presents details about each components role and the way they interact.

### Metadata Consumer

A metadata consumer is an application, user or service accessing and using metadata. The different types of Information Producers can also be metadata consumers. A metadata consumer uses the query interface of either the Information Service or an RDF Information Producer to extract metadata. There are two types of metadata consumers; grid members and public consumers. The only difference is that a grid member holds a valid AstroGrid-D certificate.

### Information Service Storage

Reliable storage of the aggregate metadata in AstroGrid-D is provided by the information service storage. Metadata is represented with the Resource Description Framework (RDF) [5, 22, 25] and metadata extraction or querying is done using the SPARQL Protocol and RDF Query Language (SPARQL, pronounced "sparkle") [27]. RDF is a powerful way of describing

metadata and allows for a simple internal data model based on three value tuples. Unlike the relational data base model, changes to metadata specific semantics and schemes does not affect the design of the internal model. Thus, using RDF provides extensibility of metadata schemes and vocabularies. RDF also has the advantage of being a W3C recommendation which should admit increased interoperability. It is also easy to represent RDF in both graphic and text formats such as XML, which is useful for a more flexible metadata exchange. SPARQL is a relatively simple but powerful query language supporting the AstroGrid-D requirements. A protocol for sending SPARQL queries and returning query results is also defined [15]. The web service interface for an AstroGrid-D Information Service supports both information management, such as adding, removing and handling security for RDF metadata, and metadata querying through the SPARQL protocol.

### Information Producers

An Information Producer actively collects, transforms, aggregates or creates metadata. It can be anything from a sensor at a compute node monitoring the current CPU usage or a job management service to a researcher adding annotations to scientific data. Metadata is made available for discovery by RDF consumers by letting the Information Service store the metadata or by implementing the query interface at the actual Information Producer. Three types of Information Producers are identified.

**RDF Information Producer** This type of Information Producer stores its local metadata using an RDF store as backend and supplies RDF directly to the Information Service. RDF consumers can access the local metadata through the query interface in exactly the same way as via the Information Service, which implies that the SPARQL query protocol must be implemented. An RDF Information Producer is often part of another resource or service with functionality represented via the logic interface. An AstroGrid-D information service storage can be seen as an RDF Information Producer since metadata can be extracted via the query interface and because it has a logic interface which corresponds to the metadata management interface of the information service storage.

**Simple RDF Information Producer** The main difference between a Simple RDF Information Producer and an RDF Information Producer is that the former does not expose its local metadata through the query interface. Additionally, the logic interface may not export any public methods. This type of information producer can represent temporary producers, for example a researcher adding annotations to a data set.

**Information Producer** A non-RDF Information Producer needs to transform the produced metadata into RDF to make it available via the Information Service. Its native metadata format can, for example, be XML or a plain list with (key, attribute)-pairs.

### Security

Security is not shown in figure 6.2, but is still a vital component of the architecture. The security component part of the Information Service covers lower level service authorization and authentication, but also higher level metadata access control and differentiation between

AstroGrid-D metadata consumers and non-AstroGrid-D consumers. Section 6.3 describes security-related issues.

### Interaction between components

A component uses the public SOAP- (or REST- if available) interface of other components to communicate metadata, requests and replies. This implies the use of HTTP and XML to transport and represent data. RDF-based metadata can be serialized in a number of ways. At least RDF/XML [5] must be supported for interoperability reasons, although it should be noted that other formats should be implemented by the Information Service if necessary.

Adding metadata to the Information Service can only be done by using the information management interface. In that sense the Information Service is passive, only storing the submitted metadata. The information producers are active in the sense that they collect, create or aggregate metadata from various sources before adding it to the Information Service. Data extraction from an RDF Information Producer and an Information Service can be done through two mechanisms, query or publish/subscribe.

**Query** This method is pull-based and initiated by a consumer. The query can be a one time request or sub-sequent depending on the consumer demands.

**Publish/Subscribe** In this communication pattern, a producer publish changed information to one or more consumers. These consumers have explicitly subscribed to the producers information. The publish/subscribe model is typically used to increase scalability since it is more economical than if a consumer periodically queries the information producer.

It is important to support both mechanisms since the query mechanism can support more complex specification of information and is better suitable for single queries. Publish/subscribe should mainly be supported for scalability reasons, but also since it fits well in the loosely coupled distributed environment where the Information Service and RDF Information Producers will be deployed. A requirement to implement publish/subscribe is that a notification-mechanism for add and remove of RDF statments is supported by the RDF store. Subscription handling should be implemented by an external service and is therefore not part of the Information Service nor the RDF Information Producer interface.

### Information transformation and aggregation

Metadata produced by the different Information Producers may need special treatment such as translation or augmentation with extra information. Since the Information Service storage interfaces should be kept as simple as possible these additional service interfaces are stand-alone.

**Translator** This service transforms input data from one metadata format (i.e. XML, ...) into RDF. It can be seen as a Simple RDF Information Producer since it does not provide any query interface. A translator can also translate from RDF into another format such as HTML or another XML representation. The transformation itself can be done via,

for example, XSLT [24]. A translator must support a method for performing the actual transformation. Translated metadata is returned to the translation requester.

**Aggregator** An aggregator receives metadata from one or more RDF producers. This metadata is then forwarded to an Information Service or another aggregator. The aggregator can enforce different policies on the aggregated metadata such as a Time-to-live (TTL) or update of the value in statements. The policy functionality is not part of the Information Service storage interface since these interfaces are different depending on policy. The interface for an aggregator should at least include register metadata source and un-register metadata source. For more efficient metadata aggregation publish/subscribe should be used if available. Furthermore, an aggregator can either be a simple RDF Information Producer or an RDF Information Producer depending on whether RDF data is cached or not. When an aggregator is an RDF Information Producer it must export the Information Service query interface. This can be used to relieve the Information Service storage from query load by allowing queries directly to the aggregator.

### 6.2.3 Storage mechanisms

The following storage mechanisms represent both RDF specific and more data base specific requirements on the Information Service storage. Some of the mechanisms presented are not yet well-defined and are therefore not planned in the initial version of the Information Service storage.

#### Add and remove metadata

Add and remove are together with metadata querying the most basic functionality for the Information Service storage. RDF metadata is added and removed by using the interface specified in section 7. The input is an RDF graph serialized in any of the formats supported by the storage. Initially, this should at least include RDF/XML. Other formats should be added for performance reasons and increased flexibility. RDF/XML is mainly meant to be machine-readable unlike XML-formats in general and, since it is XML, there may be transfer and processing overhead compared to other formats such as turtle [6] or TriX [13]. Removing metadata is done in a similar fashion, by submitting a set of serialized RDF statements.

#### Query

Extraction of metadata from the storage is done by issuing queries. A query is a string defined by SPARQL query language. Submission of queries to the storage is done via the SPARQL protocol [15]. An important addition to this protocol that must be supported is AstroGrid-D certificates for authentication and authorization. When deploying a distributed storage it may be necessary to support parallel queries or query forwarding. Parallel queries, a set of queries are distributed to multiple data sources which returns the aggregated results back to the query originator. Query forwarding, in this case, can occur when an RDF statement refers to another RDF graph with additional information. A decision can then be taken to either follow the indirection or not. These and related techniques need to be evaluated for usage in a distributed Information Service storage.

## Context

The RDF model represents metadata in (subject, predicate, object)-tuples. A context (or named graph [12]) provides a way to group statements together. The context is typically represented with a URI reference and instead of triples the store must support quads (context, subject, predicate, object). Since a URI reference also can be an RDF subject it is possible to make RDF statements about the context. Within AstroGrid-D, contexts are a practical solution to implement provenance (metadata owner), fine-grained security, timestamps, context-policy, etc. .

## Schema validation

Schema validation is used to ensure that the added metadata has the correct type. For example,  $\pi$  is of type double and not of type integer. This type of validation needs to be done whenever new metadata is added to the storage. By using validation there may be a minor performance penalty, but the advantage of increased metadata quality compared to a storage without validation is more important. Validation is limited to controlling simple datatypes such as integer, string, double, boolean and date similar to the types used in [27, 8].

## Metadata life-time

The metadata life-time in AstroGrid-D can range from minutes to years. Resource metadata such as current job queue length becomes obsolete within minutes (or even seconds) while scientific annotations can have a life-time of several years or more. Instead of forcing the metadata owner to remove obsolete metadata, a time-to-live (TTL) can be specified for metadata contexts. When the TTL expires the metadata entry is removed. This mechanism can be realized by the metadata storage or in an external aggregator that keeps track of a context's TTL. It should not be obligatory to assign a TTL to a context, since some metadata, for example scientific annotations, will reside within the storage for very long time periods.

## Metadata persistence

Durability, D in the classic database term ACID, of metadata should be interpreted in such a way that when a metadata producer has received a positive notification for the addition of a metadata entry, this entry will remain in the metadata storage until explicitly removed. Entries should be retrievable even after a re-start following a system failure. It is therefore necessary to store the metadata in a persistent memory such as magnetic storage (hard disk, ...). As implied in the metadata life-time discussion, some metadata stored by the Information Service is not reproducible as easily as resource metadata. Therefore, it is vital that the Information Service also support this metadata and not only provide functionality similar to a cache.

## Publish/subscribe

A publish/subscribe-implementation must have storage support in the form of an event framework that triggers on addition or removal statements. The actual publish and subscription

handling can be implemented directly as part of the service or as a stand-alone service. The interface for publish/subscribe is not defined by the Information Service, instead it is expected to be common functionality provided by AstroGrid-D services.

### **Replication of metadata**

By replicating metadata it is possible to increase the reliability of the storage. It can also be used for load-balancing queries for better query request scalability. The most important issue with replication is consistency. In AstroGrid-D the requirements neither explicitly nor implicitly state a need for strong consistency. Updating of replicas may therefore be done through publish/subscribe. The exception is when security related statements are updated. In this case, a consistency protocol suitable for ensuring the required level of consistency needs to be deployed.

For automated replication a common software has to be pre-deployed on all nodes hosting replicas. This common software should handle the replication procedure, including replication of storage backend logic, service logic and the actual data. This functionality can also be used to safely update the software of the replica host on-the-fly.

### **Storage Partitioning**

A storage can be partitioned into smaller pieces and distributed across nodes with different geographical placement. Storage partitioning is mainly used for scalability reasons related to the amount of stored metadata and querying. It is also possible to combine partitioning and replication to achieve higher availability and reliability of the service. The biggest disadvantage with partitioning is that querying becomes more complex. Either parallel queries to all nodes hosting partitions or distributed indices must be deployed. Trade-offs between number of total nodes and query capabilities should be assessed before deciding the mechanisms to use for querying a partitioned storage. How to actually partition an RDF store needs to be examined further.

### **Hierarchies**

Since a context is represented with a URI reference it is possible to produce an explicit hierarchy of contexts. The root of the hierarchy can for example be defined with the URI reference <http://www.example.org/resources/clusterX/>. This context could include static metadata for `clusterX`. Subsequent measurements with dynamic resource metadata could have a context with URI reference <http://www.example.org/resources/clusterX/Y>, where Y is increased for each new measurement. Unlike most hierarchical structures, properties stated at a higher level is not applied to the lower levels. For example, if owner A owns `clusterX` it is not safe to assume that A also owns a context created "under" `clusterX`. The reason for this is simply that a context name does not imply hierarchy in the storage itself, i.e. all context names have equal value independent of their URI references.

### 6.2.4 Summary

The design of an Information Service providing storage and querying of the different metadata classes within AstroGrid-D was presented. A generic framework called RDF will be used for representing metadata. RDF allows for interoperability since it is an open format. Additionally, it allows for extensibility since metadata can be stored independently of storage internal schemes, unlike relational databases. Dynamic metadata and static long-lived metadata is handled by the TTL- and metadata persistence mechanisms. To increase reliability and scalability, techniques such as replication, parallel querying, storage distribution through partitioning of the Information Service storage will be used.

## 6.3 Security Model

Security is an important requirement from the users of AstroGrid-D. Users need fine-grained security with a possibility to assign different access levels to their metadata entries. An example is that in some cases only the researcher who has submitted a job should be able to see the input parameters. Another requirement is that researchers should be able to make metadata publicly available to other users of AstroGrid-D and to metadata requestors without AstroGrid-D certificates. An AstroGrid-D compatible Information Service should support the possibility to assign security relations at the metadata entry-level, i.e. fine-grained security.

The security model presented here is based on two common techniques of expressing fine-grained security, access control list (ACL) and security levels [20]. ACLs are used to assign specific users or groups privileges to a target, while security levels provide a labeling of the target that overrides the ACL. The following part of this section discusses how this can be applied to RDF, which is the metadata representation used within AstroGrid-D.

Additionally, each RDF statement has an owner. This owner must have a valid AstroGrid-D certificate, implying that no one other than AstroGrid-D users, services or resources can create data in the Information Service storage. Statements should therefore be trusted and in cases where a statement's validity is questionable it is possible to trace it back to the originator. Furthermore, this is also a way to enforce a higher quality of the metadata stored in the Information Service storage, since there will always be someone responsible for the stored information. The default behavior is that only the owner of a metadata entry can remove the entry. Access to an entry is allowed to all certified users under the assumption that an ACL does not restrict access to the entry. Due to the dynamics of users there must be a method for transferring ownership from one AstroGrid-D user to another.

Since the Information Service provides a grid service, the aspects of service level authentication and authorization also have to be considered. The Information Service requires that delegation of security credentials be enabled at the messaging layer or by using sessions. This is needed for proper request authorization against the local security model. Another requirement on the service security layer is the need of a mechanism for extracting user or service information necessary for authorization of the requestor.

### 6.3.1 Access Control Lists

An ACL part of the Information Service is a list of users and groups related to an RDF statement. The list defines the certified entities that have access and the type of access they have to a specific RDF statement. A certified entity is something in possession of a valid AstroGrid-D certificate, for example an AstroGrid-D user, service or resource. Users and groups of certified entities are defined by a third party security service. ACLs, on the other hand, are managed through the generic interface provided by the Information Service.

According to the requirements on access control it is sufficient if an ACL defines who can access a certain metadata entry. As stated earlier, the default behaviour is to allow certified entities read access to entries without ACLs. An entry with an ACL works the other way around and only allows entities specified in the ACL and the owner to access the entry. Extending ACLs to also support write access or any other privileges is straight-forward when using RDF to represent an ACL. In some cases an entity part of the ACL can be a pre-defined group of users. In those cases all users who are part of the group should have the privileges defined in the ACL. A group cannot contain other groups.

### 6.3.2 Evaluating an ACL

The access restrictions implied by ACLs have to be controlled during access to the metadata. Since an ACL is defined on the entry-level, i.e for each RDF statement, the ACL for each RDF statement have to be compared against the accessing user. The main practical issue when using ACLs is that the performance can be degraded significantly. It is foreseen that ACLs can be evaluated in three different ways.

**Query-modification** Each query is modified to contain patterns matching the user against each ACL. In some cases this can also be an optimization since the number of statements a user is allowed to access might be smaller than the number of statements in the original query.

**Post-filtering** The query is executed in a normal fashion and returns the results. All results have to be augmented to contain the ACL from the statement from which they were derived. The result is then filtered and returned to the accessing user.

**Statement access** This is the most low-level control and implies changes to the storage internals. For each access to a statement the accessing user is compared to the ACL accompanying the statement.

How these different ways of fine-grained user authorization affect the performance should be evaluated. Another important issue that is related to performance is how to keep the query results consistent if an ACL is changed during an ongoing query. One solution to this is to lock down the storage implying that only the query has access to it. These and related issues need to be evaluated to find a good trade-off between security and performance.

### 6.3.3 Security Levels

Apart from ACLs, two different security levels can be associated with a set of statements. The two levels represent public access and AstroGrid-D access only. More precisely, the public access level is the only level that can be associated with a statement. If a statement is not associated with any level then it can only be accessed by AstroGrid-D certified entities. When a statement has a public level it is part of a set of public statements that is available for search by entities without a valid certificate. Evaluation of security levels is much less restrictive than ACLs and will therefore not have as large impact on the performance.

## 7 Interfaces

The previous sections have covered the architecture, user and grid requirements and the techniques used to satisfy these requirements. This section presents the public interfaces for the Information Service storage, the translator and the aggregator components. These are the remote interfaces of the respective components and should be implemented using WSDL and SOAP. Additionally, the SPARQL query protocol, which is used by both the Information Service storage and an RDF Information Producer for metadata discovery, is introduced briefly. User authentication and authorization for usage of the service is not part of the service interfaces since it is handled through a common grid mechanism.

### 7.1 Interface description

The signature of the methods is given in the form: <method name>([<input type> <input argument name>]) throws [<exception>]: <return type>. If a method does not throw an exception then throws [<exception>] is omitted. A list is a tuple within square brackets [<list>], where the notation for a tuple is a list of <datatype name> enclosed within parenthesis (<datatype name>, ..., <datatype name>). The keyword **optional** is used to state that the method argument is optional.

### 7.2 Datatypes

Datatypes in AstroGrid-D range from UNICODE strings to more complex XML based return types. Table 7.2 defines the datatypes passed to and returned by the Information Service components.

Datatype	Description
String	A string in UNICODE format.
URI	A URI as defined in [25] with the thereby following restrictions. Note that these restrictions may or may not apply when a URI is mentioned in other parts of this document.
(URI, URI, String)	Represents an RDF triple [25], can also be written (s,p,o) which stands for (subject, predicate, object).
RDF/XML	A string serialized according to the format specified in [5].
SPARQLXML	Consist of a string formatted according to the format specified in [7].

Table 7.1: Datatypes

## 7.3 Exceptions

Exceptions are used to report errors back to a client. Table 7.2 defines the exceptions for the AstroGrid-D Information Service.

Exception name	Description
ParseException	A <code>ParseException</code> is thrown when a supplied statement had errors. Includes additional information about where the parsing failed.
StorageException	The <code>StorageException</code> is used to return information about an error when trying to add or remove statements from the storage.
StatementNotFoundException	Thrown when the given statement could not be found during, for example, remove or when changing privileges. Includes a list of all statements that could not be found.
FileNotFoundException	Thrown when the file could not be accessed, either because it was not available or because the user did not have read access rights.
MalformedQuery	This exception is thrown when an error occurs during a lookup due to a malformed query.
QueryRequestRefused	A query request refused can be thrown under various conditions when querying an Information Service storage implementation. More information below and in [15].
AuthorizationFailedException	This exception is thrown when a user does not have sufficient rights for the current access.
TranslationFailedException	This exception is thrown when a translator service fails to translate the given input.

Table 7.2: List of exceptions

### Information management

The interfaces for information management provide methods to add and remove metadata to the Information Service storage.

```
void add(RDF/XML RDFData, URI context) throws StorageException, AuthorizationFailedException
```

**Parameters:**

RDFData An RDF string serialized as RDF/XML.  
 context The context where the RDFData should be added.

**Throws:** StorageException, AuthorizationFailedException

**Returns:** void

Method description: Add all statements from the RDFData string serialized as RDF/XML. The statements are added to the given `context`. An RDF Schema (RDFS) can also be added via this method since RDFS is represented as an RDF graph. Note that RDFS statements that

are part of the storage currently does not have any effect on other statements or on statements added via the `add` method. If the context does not exist, then a new context with the URI reference `context` is created. If a context with the same name already exists then any new statements are appended to the context. Throws a `StorageException` if it was not possible to add the statement and an `AuthorizationFailedException` if the user was not authorized for adding the statements in the `RDFData` argument.

```
void remove([[s,p,o]] RDFData, URI context) throws AuthorizationFailedException
```

**Parameters:**

`RDFData` A list of RDF statements in the form (subject, predicate, object).  
`context` The context from which the statements should be removed.

**Throws:** `AuthorizationFailedException`

**Returns:** `void`

Method description: Removes the RDF statements from the given context. Throws a `AuthorizationFailedException` the requester does not have the valid credentials for removing any of the statements. If this exception is thrown then none of the statements in the list will be removed even if the requester is authorized to remove part of the statements in the list.

```
void update(RDF/XML RDFData, URI context) throws AuthorizationFailedException
```

**Parameters:**

`RDFData` An RDF string serialized as RDF/XML.  
`context` The context from which the statements should be removed.

**Throws:** `AuthorizationFailedException`

**Returns:** `void`

Method description: Updates RDF statements from the given context. Throws a `AuthorizationFailedException` the requester does not have the valid credentials for updating any of the statements. If this exception is thrown then none of the statements in the list will be updated even if the requester is authorized to update part of the statements in the list. The update method matches the (subject, predicate) and replaces the object in matching statements. Additional statements in `RDFData` are added to the storage.

## Security management

void **allowReadAccess**(URI context, URI userOrGroupID) **throws** AuthorizationFailedException

**Parameters:**

**context** A context on which the given user id or group id should be allowed read access.  
**userOrGroupID** A resource representing a user or group.  
**Throws:** AuthorizationFailedException  
**Returns:** void

Method description: Enables read access on the given context to the given user id or group id. The user id or group id must be a valid AstroGrid-D certified entity (defined in section 6.3). An AuthorizationFailedException is thrown when the requester is not authorized for this operation. Only an owner can allow other users or groups read access.

RDF/XML **getACL**(URI context) **throws** AuthorizationFailedException

**Parameters:**

**context** A URI reference representing a context.  
**Throws:** AuthorizationFailedException  
**Returns:** An RDF graph represented in RDF/XML.

Method description: Returns the access control list for the specified context. Throws AuthorizationFailedException if the accessing user do not have rights to retrieve the ACL for the given context.

void **setPublicLevel**(URI context) **throws** AuthorizationFailedException

**Parameters:**

**context** A URI reference representing a context.  
**Throws:** AuthorizationFailedException

Method description: Changes access level to *public* on the given context. The result of this is that for all RDF statements that are part of the context becomes accessible to the public. This overrides any current ACL assigned to the context. Throws a AuthorizationFailedException if the user is not allowed to perform this operation.

void **unsetPublicLevel**(URI context) **throws** AuthorizationFailedException

**Parameters:**

**context** A URI reference representing a context.  
**Throws:** AuthorizationFailedException

Method description: Removes the public level from the given context. Throws a AuthorizationFailedException if the user is not allowed to perform this operation.

---

**String getLevel**(URI context) **throws** AuthorizationFailedException

**Parameters:**

context A URI reference representing a context.

**Throws:** AuthorizationFailedException**Returns:** A String stating the level.

Method description: Retrieves the current level for the given context. Throws a AuthorizationFailedException if the user is not allowed to perform this operation.

**void changeOwner**(URI context, URI userID) **throws** AuthorizationFailedException

**Parameters:**

context A URI reference representing a context.

userID A URI (RDF subject) representing a user ID.

**Throws:** AuthorizationFailedException

Method description: Changes owner of a context from the current owner to the owner with the given user ID. Throws a AuthorizationFailedException if the user is not allowed to perform this operation. This can only be done by the current owner or an administrator.

## Translator

A translator service is implemented to handle a specific translation and there are no default translators. Translators should be implemented by an information producer and made available to AstroGrid-D when needed. An example of a translator could be a service transforming XML GLUE schema metadata to an RDF representation that can be stored in the Information Service storage.

**RDF/XML translate**(String input) **throws** TranslationFailed

**Parameters:**

input A string in a metadata format (XML, (key, value)-tuples, ...)

**Throws:** TranslationFailure**Returns:** RDF/XML

Method description: Translates the given input in a format known by the translator service into RDF/XML. Throws a TranslationFailedException if the translation fails.

## Aggregator

An aggregator interface is typically implemented as part of another service, but can also be implemented stand-alone. Aggregators provide a simple way to concentrate site-information

output to a single source.

```
void register(URI resource, integer TTL) throws
```

**Parameters:**

resource An information source represented as a URI.  
TTL A value for time-to-live.

Method description: Registers a new metadata source which should be contacted via the given resource URI. It will be removed after TTL seconds if not updated.

```
void unregister(URI resource) throws
```

**Parameters:**

resource An information source represented as a URI.

Method description: Unregisters a source of information. This can only be done by the source itself.

## Query

The query interface complies with the query interface defined for the SPARQL query language as described in [15, 7]. This query interface exports one method, `query`, defined below using this document's interface conventions with minor modifications. This method has two types of return values that depend on the submitted query. The HTTP interface defined in [15] may not be implemented if authentication and authorization cannot be implemented in accordance to the security protocol deployed within AstroGrid-D.

```
SPARQLXML, RDF/XML query(String query) throws MalformedQuery, QueryRequestRefused
```

**Parameters:**

input A string representing a SPARQL query as defined in [27].

**Throws:** QueryRequestRefused

**Throws:** MalformedQuery

**Returns:** RDF/XML

**Returns:** SPARQLXML

Method description: Performs `query` on the RDF metadata stored in the Information Service storage. The results are serialized as RDF/XML or in the SPARQLXML format depending on the type of query. Statements in a query result for which the requester is not authorized are not returned. Further details and specification of the exceptions is found in the SPARQL protocol for RDF specification and the SPARQL Query Results XML format specification.

## 8 Scenarios

In this chapter different scenarios are presented that gives a brief introduction in what different ways the Information Service can be used from both the grid and the user perspective. These scenarios cover the most important aspects of the Information Service; insertion of new metadata, discovery of existing metadata, access to public or access restricted metadata, change access restrictions to metadata and updating dynamic resource information. It should be noted that the scenarios are often hiding details to make the scenarios simpler. Details of the Information Service are discussed in the architecture chapter 6.

The figures show the different actors that are part of a scenario in boxes extended with lines representing time. A flow of event is a set of arrows indicating the direction of the message or request between the actors. A dashed line represent an alternative flow of events, for example an error. When an alternative flow of events occur before the primary flow of events, the latter is interrupted. Arrows going back to the same actor should be interpreted as an internal method call. Additional data passed during a request is denoted under the request line. This can for example be security credentials, which are not explicit as arguments part of the request.

### 8.1 Register a new grid job

Figure 8.1 shows a typical grid scenario where a job management service tries to add metadata about a new job to the Information Service. It is assumed that a user of the job management service has submitted a new job. By adding the job metadata entries to the Information Service, the job management enable other services and AstroGrid-D users to find information about this job. The job management starts with calling the method for adding metadata job information. This method generates the metadata entries to be added to the Information Service. The job management is notified if an error occurs such as authentication/authorization failure or if the Information Service storage cannot add the entries for some other reason.

### 8.2 Find all running grid jobs for a user

The main objective in this scenario, presented in figure 8.2, is to find all currently active jobs for the user. This and the next scenario is rather similar, they try to highlight that the real work is in constructing the query passed to the Information Service interface. The reasons behind this design decision is discussed more in section 6.

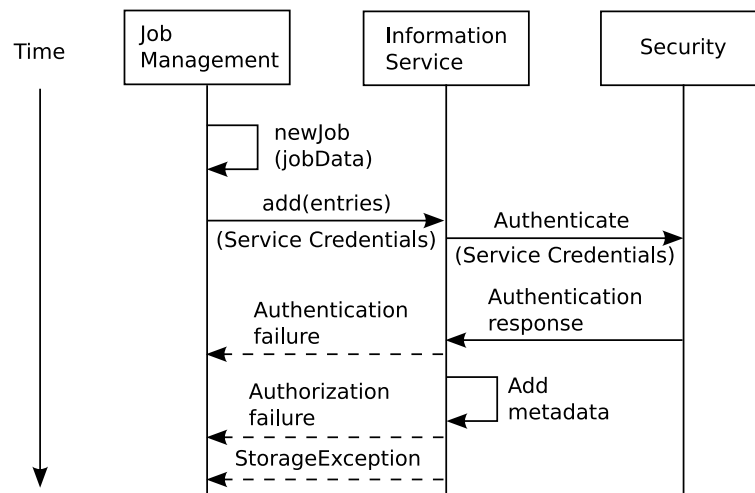


Figure 8.1: Flow of events for adding metadata about a new job.

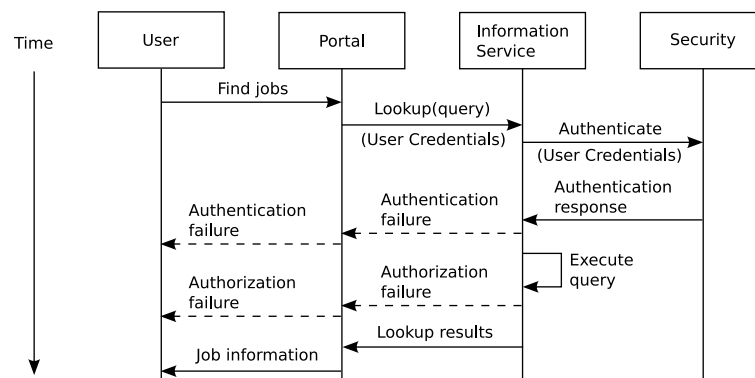


Figure 8.2: The steps that occur when a user uses the portal to find all running grid jobs for the user.

### 8.3 Access to application generated metadata

In this scenario the user tries to find metadata about a simulation that is already made available for discovery in the Information Service. A portal provides a user interface for input of the search criteria. It would also be possible for the user to contact the Information Service directly with a user specified query. This scenario focuses on how it works when the user is using a portal. The search criteria is processed by the portal and submitted to the Information Service. If the user authentication fails, this is reported back to the user which is shown as an alternative flow of events in figure 8.3. A successful user authentication leads to a reply containing the search results. It should be noted that the Information Service only returns the results for which the user is authorized. If the user does not have rights to any of the results an empty result set is returned without explicitly stating that the user was not authorized to view the results. The authorization is shown as an internal method call in the figure.

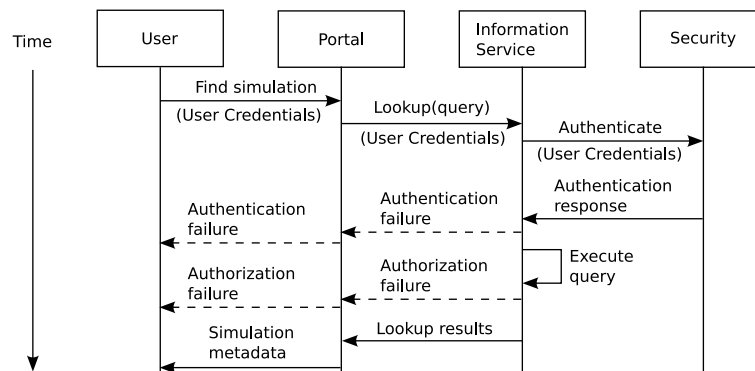


Figure 8.3: Request and response flow when finding application generated metadata.

## 8.4 Apply security information to metadata

In this scenario the user is interacting directly with the Information Service via its exported interface part of the AstroGrid-D middleware. The objective is to allow another user to read the information in a metadata entry stored as part of the Information Service. Figure 8.4 gives the detailed flow of events for this scenario. A failure of applying the security information results in an explicit return message stating the cause of failure. Nothing is returned on success.

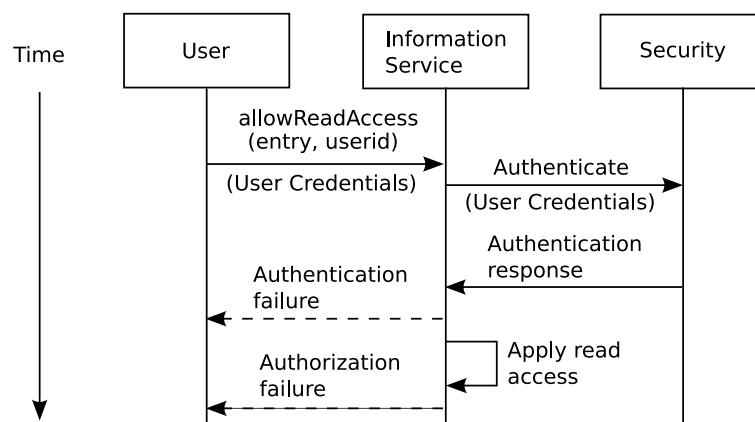


Figure 8.4: Flow of events when a user tries to add read access to another user for a metadata entry.

## 8.5 Update dynamic resource information

Figure 8.5 shows the flow of events when a resource monitor, using metadata from the GLUE schema, tries to update its dynamic resource information (size of job queue, CPU usage, ...). Since the information service storage does not understand metadata from the GLUE schema, this information has to be translated into RDF. The translation itself is done via a translator

service. For this scenario, the translation mechanism from the GLUE schema in XML to the RDF GLUE schema was pre-loaded. There are a number of different failures for this scenario. First, authentication and authorization failures can occur at both the information service and the translator service. Additionally, the translator can fail when the submitted data is not valid GLUE schema XML. The information service may return an error if it was not possible to store the submitted GLUE schema in RDF format.

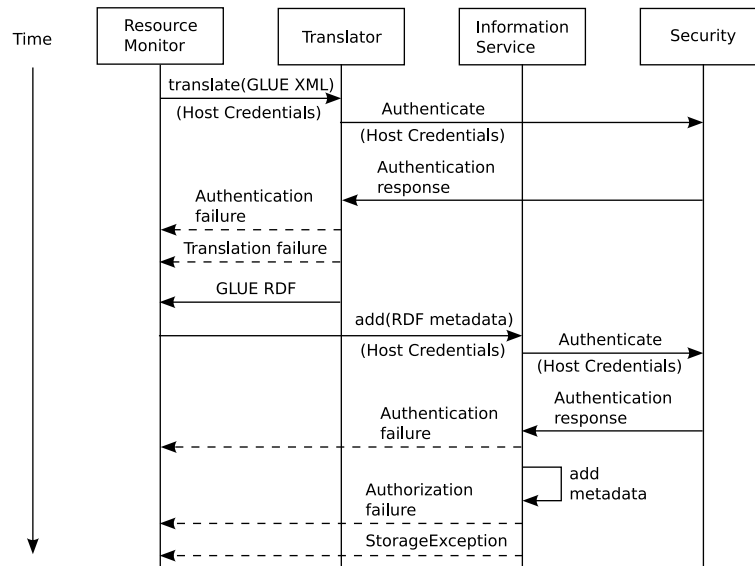


Figure 8.5: Flow of events when a resource monitor updates dynamic resource information.

## **F: References / Bibliography**

## Bibliography

- [1] Enabling grids for e-science. <http://egee-intranet.web.cern.ch/egee-intranet/index.html>.
- [2] The globus foundation. <http://www.globus.org/>.
- [3] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Konya, M. Mambelli, J. M. Schopf, M. Viljoen, and A. Wilson. Glue schema specification version 1.2. Technical report, December 2005.
- [4] Sergio Andreozzi, Natascia De Bortoli, Sergio Fantinel, Antonia Ghiselli, Gian Luca Rubini, Gennaro Tortone, and Maria Cristina Vistoli. Gridice: a monitoring service for grid systems. *Future Generation Comp. Syst.*, 21(4):559–571, 2005.
- [5] D. Beckett. Rdf/xml syntax specification (revised). <http://www.w3.org/TR/rdf-syntax-grammar/>, February 2004.
- [6] D. Beckett. Turtle - terse RDF triple language. <http://www.dajobe.org/2004/01/turtle/>, April 2006.
- [7] D. Beckett and J. Broekstra. SPARQL query results XML format. <http://www.w3.org/2001/sw/DataAccess/rf1/>, January 2006.
- [8] P. V. Biron and A. Malhotra. XML schema part 2: Datatypes second edition. <http://www.w3.org/TR/xmlschema-2/>, October 2004.
- [9] DCMI Usage Board. Dcmi metadata terms, (dublin core). <http://dublincore.org/documents/dcmi-terms/>, June 2005.
- [10] D. Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF schema. <http://www.w3.org/TR/rdf-schema/>, February 2004.
- [11] D. Brickley and L. Miller. FOAF vocabulary specification. <http://xmlns.com/foaf/0.1/>, July 2005.
- [12] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM Press.
- [13] J. J. Carroll and P. Stickler. RDF triples in XML. Technical Report HPL-2003-268, HP Labs, June 2004.
- [14] R. Chinnici, J-J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (WSDL) version 2.0 part 1: Core language. <http://www.w3.org/TR/wsd120/>, March 2006.

- 
- [15] E. G. Clark. SPARQL protocol for RDF. <http://www.w3.org/TR/rdf-sparql-protocol/>, January 2006.
- [16] Ewa Deelman, Gurmeet Singh Singh, Malcolm P. Atkinson, Ann L. Chervenak, Neil P. Chue Hong, Carl Kesselman, Sonal Patil, Laura Pearlman, and Mei-Hui Su. Grid-based metadata services. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 393–402. IEEE Computer Society, June 2004.
- [17] S. Derriere, N. Gray, R. Mann, A. P. Martinez, J. McDowell, T. McGlynn, F. Ochsenbein, P. Osuna, G. Rixon, and R. Williams. An IVOA standard for unified content descriptors version 1.10. <http://www.ivoa.net/Documents/latest/UCD.html>, August 2005.
- [18] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
- [19] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana. Modeling stateful resources with web services v. 1.1. <http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf>, March 2004.
- [20] D. Gollmann. *Computer Security*. John Wiley and sons, 2002.
- [21] M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, and H. F. Nielsen. SOAP version 1.2 part 1: Messaging framework. <http://www.w3.org/TR/soap12-part1/>, June 2003.
- [22] P. Hayes. RDF semantics. <http://www.w3.org/TR/rdf-mt/>, February 2004.
- [23] F.V. Hessman, C. Pennypacker, E. Romero-Colmenero, and G. Tuparev. Telescope networking and user support via remote telescope markup language. In *Advanced Software, Control and Communication Systems for Astronomy, SPIE*, pages 290–301, 2004.
- [24] M. Kay. XSL transformations (XSLT) version 2.0. <http://www.w3.org/TR/wsd120/>, November 2005.
- [25] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. <http://www.w3.org/TR/rdf-concepts/>, February 2004.
- [26] F. Manola and E. Miller. RDF primer. <http://www.w3.org/TR/rdf-primer/>, February 2004.
- [27] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. <http://www.w3.org/TR/rdf-sparql-query/>, February 2006.
- [28] N. Santos and B. Koblitz. Metadata services on the grid. In *Proceedings of Advanced Computing and Analysis Techniques (ACAT'05)*, May 2005.
- [29] M. K. Smith, C. Welty, and D. L. McGuinness. OWL web ontology language guide. <http://www.w3.org/TR/owl-guide/>, February 2004.
- [30] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, and M. Swany. A grid monitoring architecture. Technical Report GWD-PERF-16-2, Global Grid Forum, January 2002.

- [31] A. J. Wilson, R. Byrom, L. A. Cornwall, M. S. Craig, A. Djaoui, S. M. Fisher, S. Hicks, R. P. Middleton, J. A. Walk, A. Cooke, A. J. G. Gray, W. Nutt, J. Magowan, P. Taylor, J. Leake, R. Cordenonsi, N. Podhorszki, B. Coghlan, S. Kenny, O. Lyttleton, and D. O'Callaghan. Information and monitoring services within a grid environment. In *Computing in High Energy and Nuclear Physics (CHEP)*, September 2004.