

A Metadata Catalog Service for Data Intensive Applications

Ann Chervenak, Ewa Deelman, Carl Kesselman, Laura Pearlman, Gurmeet Singh

Version 1.0

1 Introduction

The term *data grid* refers to computational grid middleware designed to support data-intensive, high-performance computing applications. Examples of data-intensive applications include experimental analyses and simulations in scientific disciplines such as high-energy physics, gravitational-wave physics, climate modeling, earthquake engineering, and astronomy. In these applications, massive datasets are shared by a community of hundreds or thousands of researchers around the world. Fundamental data grid services include efficient data transport, replica management and the management of metadata associated with the datasets.

In this paper, we present a design for a prototype Metadata Service for data grids. The Metadata Service provides a mechanism for storing and accessing metadata, which is information that describes data files or data items. The Metadata Service (MCS) allows users to query based on attributes of data rather than data names. In addition, the MCS provides management of logical collections of files and containers that consist of small files that are stored, moved and replicated together.

There are various types of metadata. In our prototype service, we distinguish between logical file metadata and physical file metadata. By design, our metadata service exclusively contains information that describes logical files. In our data model, the term *logical file name (LFN)* denotes a unique logical identifier for data content. There may be zero or more physical replicas of the content on storage systems, each of which is uniquely specified by a *physical file name (PFN)*. We assume that physical file metadata (which depends on the actual location of the file and the characteristics of a given storage system) is stored elsewhere, most frequently on the storage or file systems where files are stored or in replica location services.

Examples of logical file metadata include:

8/5/2002

- information about how data files were created or modified (e.g., the creator or modifier, the creation or modification time, what experimental apparatus and input conditions produced the file, or what simulation or analysis software was run on which computational engine with which input parameters)
- description of what the data stored in a file represent (e.g., precipitation measurements over South America for December 1998 or particle collisions in the Large Hadron Collider for a period of 1 second)
- file format information (e.g., netCDF vs. XML vs. ASCII vs. binary).

The design of our Metadata Service is based to a large extent on the MCAT Metadata Catalog of the Storage Resource Broker from the San Diego Supercomputing Center [1]. Many of the attributes and ideas in this paper have direct parallels in MCAT. For example, both MCAT and our Metadata Service support a logical name space that is independent of physical name space; both provide GSI authentication; both allow specification of a logical collection hierarchy; and both support the notion of containers to aggregate small files. However, our catalog differs from MCAT in significant ways. First, we restrict our metadata service to holding metadata that describes files, while MCAT also stores metadata for resources, users and methods. Second, as already mentioned, we do not include metadata related to physical file properties in our Metadata Service. In addition, MCAT is tightly integrated with the Storage Resource Broker and uses a proprietary data exchange format to communicate with the SRB server, while our proposed implementation is a stand-alone Metadata Service that uses a standard XML interface to communicate with the client. Finally, the SRB implementation depends on a commercial database back-end such as Oracle or DB2, while our initial implementation will use an open source MySQL back-end.

2 An Example of the Use of a Metadata Service

Many of the applications targeted in this work publish data as a community; for example when results of a given experiment or series of experiments are available, they are calibrated in some fashion, put in a standard format, and made available (published) to the community. Some members of the community might wish to annotate the data with their own observations or produce new data products using different analysis. Other members of the community will use the data products in their published form; however, they might want to organized the data of interest to them in a convenient form (a customized view).

Figure 1 illustrates the simplest scenario for attribute-based data access via the MCS and the other components of the data grid, including the Replica Location Service and the particular storage system where the data resides.

1. The client application (or a request planner tasked with satisfying a client's request) queries the Metadata Service based on some attributes of the desired data.

8/5/2002

2. The MCS returns the logical file names of one or more data items corresponding to those attributes.
3. The client application next queries the Replica Location Service [2] with the logical file names.
4. The RLS returns the physical file names of the requested files to the client application.
5. The client application then contacts the physical storage system where the file resides.
6. The file is returned by the physical storage system.

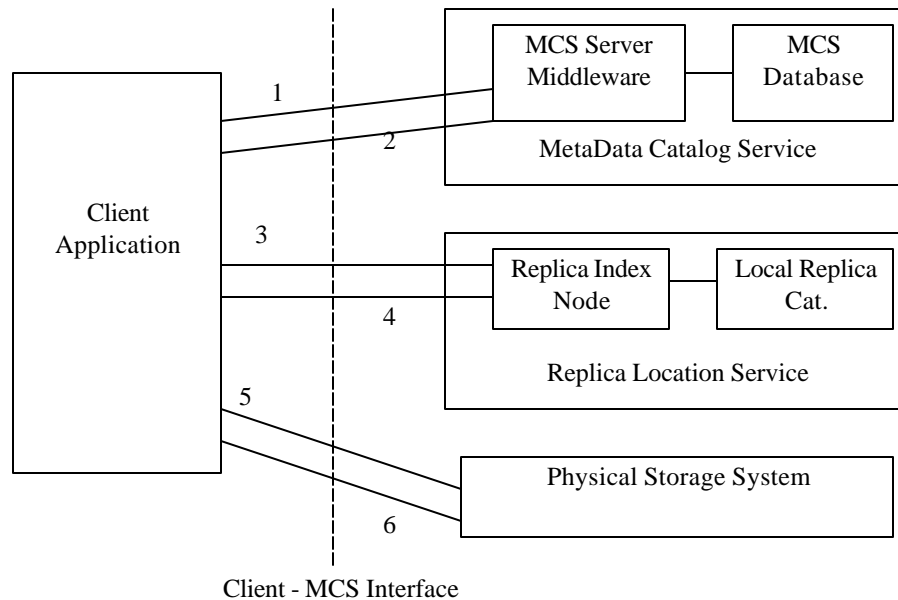


Fig 1. A usage scenario of the MetaData Catalog Service

3 Requirements for the Metadata Service

The Metadata Service must meet the following requirements:

- *Storing attributes:* The MCS must provide a mechanism for storing various metadata attributes associated with logical files.
- *Querying:* In response to queries about metadata attributes, the MCS must return the names of all logical files that possess those attributes. In addition, the MCS must respond to queries about one or more attributes of a logical file.

8/5/2002

- *Extensibility*: The MCS must be extensible. This includes support for user-defined attributes as well as the capability to access application-specific metadata catalogs external to the Metadata Service.
- *Consistency*: The MCS must maintain strict consistency over its contents. In particular, if the MCS is replicated, then all copies of the metadata database must be updated atomically. Inaccuracies in the metadata database would cause incorrect identification of data items, resulting in incorrect analyses.
- *Support for authentication and authorization*: The MCS must provide authentication based on the Grid Security Infrastructure (GSI) [3]. The Metadata Service must provide authorization/access control based on attributes stored in the metadata service as well as community policies specified in the Community Authorization Service (CAS) [4].
- *Support for logical collections*: A *logical collection* is a user-defined aggregation of logical files. One of the most important purposes of logical collections is to support authorization on groups of files rather than on individual files. If a logical file is contained in a logical collection, then the user must have appropriate permissions on both the logical file and the logical collection to access or modify the metadata associated with the logical file. To support consistent authorization, the MCS requires that a logical file may belong to at most one logical collection. The MCS must support a tree hierarchy of collections with well defined rules for delegation of authorization rights to child collections. The MCS must support simple queries that list the logical files in a collection and must respond to attribute-based queries on logical collections.
- *Support for logical views*: To allow more flexibility for users to group files according to their interests, the MCS must also support aggregations called *logical views*. A logical view may aggregate any acyclic selection of logical files, logical collections or other logical views. Logical files and logical collections may belong to many different logical views. Logical views do not affect authorization, which is enforced via access rights defined for logical collections and individual logical files. (Adding a file to a logical view is loosely analogous to creating a symbolic link in one file system directory to a file that actual resides in another directory.)
- *Creation information*: The MCS should record information about the creator of logical files, collections and views as well as creation times.
- *Annotations*: The MCS must allow users to add descriptive text as annotations to logical files, logical collections and logical views.
- *Audit records*: The MCS should provide the ability to log all the accesses to a particular data item, including the identity of the user and the action that was performed.
- *Transformation history*: The MCS should provide the capability to store records of transformations on a dataset, including its creation and subsequent processing. These transformation records may include the identity of data modifiers as well as information about analyses run and the input parameters used.
- *Master copy support*: The MCS must provide support for associating master copy attributes with logical files and answer queries about these attributes. The MCS must

8/5/2002

also provide a means of locating the master copy, possibly by referral to an external service, such as a replica location service.

- *Versioning:* The MCS must provide support for multiple versions of a particular logical file.
- *Support for containers:* Many applications produce large numbers of relatively small files that are inefficient to store and transfer individually. *Containers* allow these files to be grouped together and managed as a single unit for the purposes of data storage and movement. The MCS must interface with an external container management service that constructs containers and extracts individual files from within containers. The MCS should provide attributes that enable logical files to be associated with containers via a particular external container management system.
- *Performance:* The MCS should provide short latencies on query and update operations and support relatively high query and update rates. Performance requirements may vary based on applications.
- *Scalability:* The MCS should scale to support information about millions of logical files and thousands of logical collections and logical views.

4 A Metadata Service Prototype

For our initial prototype, we will implement a simple, centralized Metadata Service based on open source relational database technology. However, eventually this design will evolve to a distributed system that may be based on other database technologies. In this section, we discuss some of implementation considerations for the Metadata Service.

A centralized Metadata Service is simple and will allow us to experiment with the metadata schema in the context of several grid applications. However, the centralized approach presents obvious reliability and performance limitations. A centralized catalog represents a single point of failure. If the Metadata Service fails, then no attribute-based access to the logical files is possible. A centralized service is also likely to become a performance bottleneck in a large distributed system, with possibly many users performing query operations and authorized users performing updates.

Eventually, we plan to replicate the contents of the metadata service and possibly distribute the catalog, partitioning its contents among a number of locations. Recall that the metadata catalog must maintain strict consistency over its contents to avoid incorrect identification of data items that will lead to inaccurate analyses. Replication with strict consistency is challenging in wide area distributed environments. Any updates to the replicated service require distributed transactions, which may be slow. Because of these considerations, we expect the metadata service will eventually be replicated over a relatively small number of locations to provide fault tolerance and load balancing.

8/5/2002

Another technique that can provide load balancing is to distribute or partition the metadata service across multiple locations. Distribution of the service reduces the storage requirements and request load that must be serviced by each component. However, unless the distributed servers are also replicated, each one represents a single point of failure for the metadata it contains.

Our initial prototype will use standard relational database technology. We will also consider the use of newer technologies as they become available, including native XML databases. Such databases take advantage of the growing importance of XML as a standard data exchange format.

Figure 2 shows our proposed prototype architecture. Client applications will send requests to the Metadata Service via a client API. The Metadata Service prototype will consist of a web server front end and a relational database back end. We will initially use MySQL as our database back end and a server and API written in C. Eventually, we will evolve the prototype to use a SOAP interface and, most likely, a database grid service.

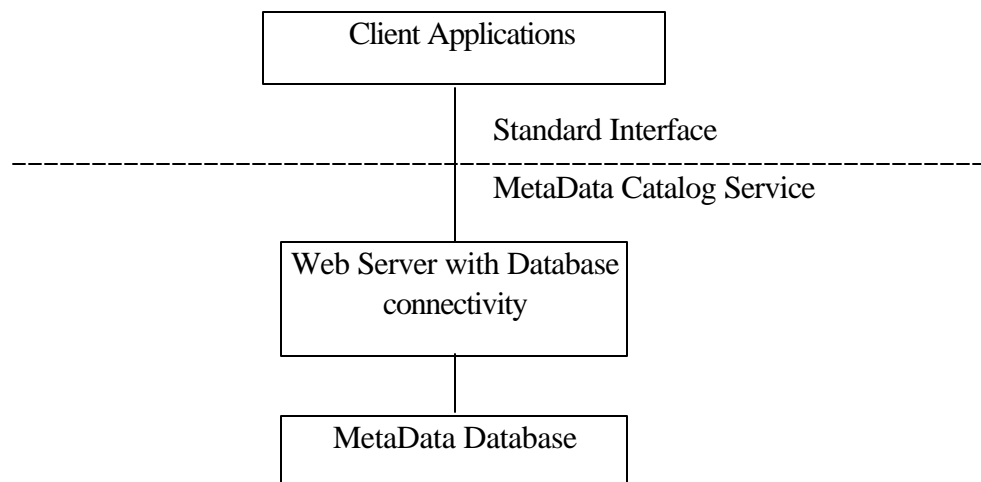


Fig 2. High Level Diagram of the MetaData Catalog Architecture

5 Proposed Schema

In this section, we present our initial schema design for the Metadata Service. The attributes in the schema can be divided into the following logical categories:

- 1) *logical file metadata*, including logical file name, version number, container information, and information about the creator and last modifier of the data.

8/5/2002

- 2) *logical collection metadata*, including collection name, the set of files that compose a collection, annotations on the collection, information about the creator and modifier(s) of the collection, and collection hierarchy information
- 3) *logical view metadata*, including the view name, view attributes, view creator and/or modifier, logical files, collections, sub views within a view.
- 4) *authorization metadata*, which is used in the absence of an external authorization service such as the Community Authorization Service to specify access privileges on logical files or collections.
- 5) *metadata that describes writers* of metadata, including contact information
- 6) *audit metadata* used to record actions performed via the metadata service
- 7) *user-defined metadata* attributes on logical files, logical collections and logical views that provide extensibility beyond pre-defined metadata attributes
- 8) *annotation metadata* that are used to describe logical files, collections, and views.
- 9) *transformation history metadata* that records information about how a logical file was created and what subsequent transformations were performed on the data.
- 10) *external catalog access metadata* that provides information needed to contact external metadata catalogs, such as existing application-specific metadata catalogs.

5.1 Logical File Metadata

The following table describes the main attributes of a logical file. The *Data_id* is the internal identifier of the data item used by the database back end, while the *Logical_name* is the logical file name of the data item. The *Data_id* field is used as a primary key for this table and for other tables that contain logical file metadata attributes. The *Data_type* attribute is a string that describes the data item type, for example whether the file format is stored as binary, html, XML, SGML, Wave Audio, or is a tar File or perl script. The *Is_Valid* attribute indicates whether a data item is currently valid, allowing us to quickly invalidate logical files, for example, if a virtual organization determines that the logical files contain incorrect data.

If data files are updated over time, then there may be multiple versions of a logical file that differ by version number. Within the metadata database, versions of a logical file are distinguished by different *Data_id* and *version* attributes. Both the logical file name and the version number must be supplied by the client to allow the metadata service to uniquely identify the desired data item. By default, the version number is one.

Collections allow aggregation of logical files and collections. The *collection_id* attribute indicates to which collection the logical file belongs. Collections also allow us to formalize the access control on groups of files. If files belong to a given collection, then they have the same access control as the collection to which they belong. Note, that in order to have a consistent authorization structure, a file can belong to at most one collection. It is also possible for a logical file not to belong to any collections, in which case the access control for the file is defined at the logical file level.

8/5/2002

Containers allow files to be grouped together and managed as a single unit for the purposes of data storage and movement. A logical file can belong to at most one container. The *Container_id* attribute records to which container a logical file belongs. The *Container_Service* provides information required to access the external service that constructed the container.

The distinguished name of the logical file entry's creator and the creation time are stored in the attributes *Writer_Dn* and *Create_Time*. Likewise, the distinguished name of the last entity to modify the logical file entry and the time of the last modification are stored in the *Last_Modifier_Dn* and *Last_Modify_Time* attributes.

We also support recording the location of a master copy of a file. Master copies may be designated to indicate a definitive copy of the file to be used during data access or as a source for replicating data. We assume that the prototype Metadata Service will be used in combination with the Replica Location Service being developed as part of the Globus toolkit. In this context, the *Master_Copy* attribute of a logical file specifies a pointer to the local catalog in the Replica Location Service that contains the mapping from the logical file name to the physical location of the master copy. At this point we support only one master copy of a file, however, if requirements dictate a multiple master solution, we will update the schema accordingly.

Finally, the *Audit* attribute indicates the type of auditing records that are maintained: no auditing; auditing for all metadata accesses; or auditing only metadata modification operations.

MCS_LOGICAL_FILE

Field Name	Type	Remarks	Description
Data_id	Integer	Non null	The data identifier
Logical_name	Varchar(250)	Non null	The logical file name
Version	Integer		The version of the data
Data_type	Varchar(250)		The type of data i.e. image etc
Collection_id	Integer		The collection to which the logical file may belong
Container_id	Integer		The container id the dataset belongs to if any
Container Service	Varchar(250)		The container management service, if any
Is_Valid	Integer	Non null	Whether the dataset is valid. Supports deletion by marking attribute
Creator_Dn	Varchar(250)	Non null	The distinguished name of the creator of the logical file metadata entry
Last_Modifier_Dn	Varchar(250)		The distinguished name of the last

8/5/2002

			modifier of the logical file metadata
Create_Time	Date/Time	Non null	The creation time of the logical file metadata entry
Last_Modify_Time	Date/Time		The time of the last modification of the logical file metadata
Master_Copy	Varchar(250)		Pointer to LRC containing the master copy of the data item
Audit	Integer	Non null	An integer with the following valid values 0 – no audit 1 – audit on for all accesses 2 – audit only modifications

5.2 Logical collection metadata

Logical collections are user-defined associations of logical files. The logical collections also allow us to build a consistent and efficient authorization structure, where authorization on files can be determined by the collection the file belongs to. Because logical collections are used for authorization on aggregates of logical files, a logical file may belong to at most one logical collection. The following table contains attributes that describe logical collections, including an internal identifier (*Collection_id*) of the logical collection used by the database back end, the name of the logical collection (*Collection_name*), and a textual description of the collection (*Collection_desc*). The distinguished name of the logical collection entry's creator and the creation time are stored in the attributes *Creator_Dn* and *Create_Time*. Likewise, the distinguished name of the last entity to modify the logical collection entry and the time of the last modification are stored in the *Last_Modifier_Dn* and *Last_Modify_Time* attributes. The *Parent_id* attribute is used to specify the parent of a logical collection. We allow an arbitrarily deep acyclic hierarchy of logical collections. The *audit* attribute records whether audit information is to be recorded for this collection.

MCS_COLLECTION

Field Name	Type	Remarks	Description
Collection_id	Integer	Non-null	The collection identifier
Collection_name	Char(250)	Non-null	The name of the collection
Collection_desc	Varchar(250)	Non-null	A description of the collection
Creator_Dn	Varchar(250)	Non-null	The distinguished name of the creator of the logical collection entry
Last_Modifier_Dn	Varchar(250)		The distinguished name of the last

8/5/2002

			modifier of the logical collection entry
Create_Time	Date/Time	Non-null	The creation time of the logical collection entry
Last_Modify_Time	Date/Time		The time of the last modification to the logical collection entry
Parent_id	Integer		The identity of the parent of this logical collection must be a valid collection_id
Audit	Integer	Non-null	An integer with the following valid values 0 – no audit 1 – audit on for all accesses 2 – audit only modifications

5.3 Logical View Metadata

A logical view is a user defined aggregation of logical files, logical collections or other logical views. The following table contains attributes that describe logical views. The *View_id* is an internal identifier used to identify the logical view by the database back end. *View_name* is the name of the logical view. Logical view names must be unique in the MCS database. *View_desc* provides a user-defined textual description of the logical view. The distinguished name of the logical view entry's creator and the creation time are stored in the attributes *Creator_Dn* and *Create_Time*. Likewise, the distinguished name of the last entity to modify the logical view entry and the time of the last modification are stored in the *Last_Modifier_Dn* and *Last_Modify_Time* attributes. The *Audit* attribute specifies the type of auditing performed for audit entries.

MCS_VIEW

Field Name	Type	Remarks	Description
View_id	Integer	Non-null	The view identifier
View_name	Char(250)	Non-null	The name of the view
View_desc	Varchar(250)	Non-null	A description of the view
Creator_Dn	Varchar(250)	Non-null	The distinguished name of the creator of the logical view entry
Last_Modifier_Dn	Varchar(250)		The distinguished name of the last modifier of the logical view entry
Create_Time	Date/Time	Non-null	The creation time of the logical view entry
Last_Modify_Time	Date/Time		The time of the last modification to the logical view entry

Audit	Integer	Non-null	An integer with the following valid values 0 – no audit 1 – audit on for all accesses 2 – audit modifications only
-------	---------	----------	---

The following table defines the mapping between logical views and the objects they aggregate (logical files, logical collections or other logical views). View must be acyclic. Logical files and logical collections may belong to multiple logical views.

MCS_VIEW_MAPPINGS

Field Name	Type	Remarks	Description
View_id	Integer	Non-null	The view identifier to which the mapping refers
Object_id	Integer	Non-null	An object contained in the view (may be a logical file, logical collection or another logical view)
Object_type	Integer	Non-null	The type of the contained object 0 – logical file 1 – logical collection 2 – logical view

5.4 Authorization Information

Authorization information is associated with both individual logical files and logical collections. The authorization information must be maintained for both individual users and a Community Authorization Service (CAS).

Logical collections allow authorization on groups of files without requiring that permissions be specified on each logical file. To support consistent authorization, the MCS requires that a logical file may belong to at most one logical collection. The access permissions for a collection apply to all logical files within the collection. In addition to the permissions specified on the collection, the user might impose additional access restrictions on individual logical files. If so, the access to the file attributes is determined by the intersection of the access permissions on the file and collection.

Some examples of the kinds of permissions that must be defined are:

- Read or query permission on logical file attributes
- Modify permission on logical file attributes
- Permission to add a logical file to MCS

8/5/2002

- Permission to add a logical file to a logical collection
- Permission to add a logical collection to a logical view

The table below associates an internal database identifier (*Permission_id*) with each set of permissions (*Permission*) such as those in the examples above.

MCS_PERMISSIONS

Field Name	Type	Remarks	Description
Permission_id	Integer	Non-null	The integer identifier for the permissions
Permissions	Varchar(250)	Non-null	The actual permissions as described above.

The next table maps the permissions defined above to particular objects and particular users. The *Object_id* and *Object_type* attributes identify the object (logical file, logical collection or logical view) on which permissions are granted. The *Perm_id* is one of the permissions defined in the previous table. The *Subject* and *Subject_type* attributes identify the entity to whom these permissions are granted. The subject entity may be an individual user or a Community Authorization Service (CAS). The permissions specified for a logical collection or logical view apply to all members of the collection or view. When multiple sets of permissions apply, the authorization is performed based on the intersection of these permissions.

MCS_DATA_PERMISSIONS

Field Name	Type	Remarks	Description
Object_id	Integer	Non-null	The id of the logical file, collection or view to which the permissions refer
Object_type	Integer	Non-null	The type of the object 0 – logical file 1 – logical collection 2 – logical view
Perm_id	Integer	Non-null	The actual permissions to which we are referring
Subject	Varchar(50)	Non-null	The name of the user (which may be an individual user or the CAS server) to whom the permissions are granted
Subject_type	Integer	Non-null	The type of the subject 0 – CAS server 1 – User DN

5.5 User information

The following table provides the information about the writers or modifiers of the logical files in the database. The attributes specify the distinguished name, description, institution, address, phone and email information for writers.

MCS_WRITER

Field Name	Type	Remarks	Description
Writer_dn	Integer	Non-null	The distinguished name of the writer
Writer_desc	Char(250)	Non-null	The description of the writer, i.e. developer, scientist, etc.
Writer_institution	Varchar(250)	Non-null	The institution to which the writer belongs
Writer_address	Varchar(250)		Address of the writer
Writer_phone	Varchar(250)		Phone number of the writer
Writer_email	Varchar(250)		Email id of the writer

5.6 Audit trials

This table contains audit records that record information about actions that can be performed on the Metadata Service. The audit record includes information about the name and type of the object upon which the action is performed, a description of that action, the user who performed the operation, and a timestamp for the operation.

MCS_AUDIT

Field Name	Type	Remarks	Description
Object_name	Integer	Non-null	The object name
Object_type	Integer	Non-null	The type of the object 0 – logical file 1 – collection 2 - views
Action_desc	Varchar(250)	Non-null	The action performed on the data item like read , modify etc
User_dn	Integer	Non-null	The modifier/reader of the data item
Timestamp	Date/time	Non-null	When the action is performed

5.7 User defined attributes

Because different application domains will have their own metadata schemas, it is important that the Metadata Service be easily and efficiently extensible to support user-defined attributes. Users may create attributes of the following types for any logical file or collection:

- String
- Integer

- Float
- Date
- Time
- Date/Time

To support efficient and complex queries on user-defined attribute types, each type is implemented using a separate database table. (The alternative is to save all attributes as strings and do conversions among different data types.)

One table maps from attribute names to identifiers and types. There is also an *object_type* associated with each attribute, since there may be attributes of the same name associated with logical files, logical collections and logical views.

MCS_ATTRIBUTES

Field Name	Type	Remarks	Description
Id	Integer	Non-null	The integer identifier for the attribute
Name	Varchar(50)	Non-null	The name of the attribute
Attribute_type	Varchar(20)	Non-null	One of the predefined types described above
Object_type	Integer	Non-null	0 – logical file 1 – collection 2 - view

The following tables are associated with different attribute types. Each attribute table associates an attribute identifier from the previous table with a particular object and an attribute value. The tables are arranged based on the type of attribute to facilitate efficient queries, for example value-based comparisons.

MCS_STRING_ATTRIBUTES

Field Name	Type	Remarks	Description
Att_id	Integer	Non-null	The attribute id
Obj_id	Integer	Non-null	The object id to which the attribute belongs
Att_value	Varchar(250)	Non-null	The string value of the attribute

MCS_INTEGER_ATTRIBUTES

Field Name	Type	Remarks	Description
Att_id	Integer	Non-null	The attribute id
Obj_id	Integer	Non-null	The object id
Att_value	Integer	Non-null	The integer value of the attribute

MCS_FLOAT_ATTRIBUTES

Field Name	Type	Remarks	Description
Att_id	Integer	Non-null	The attribute id
Obj_id	Integer	Non-null	The object id
Att_value	Float	Non-null	The floating point value of the attribute

MCS_DATE_ATTRIBUTES

Field Name	Type	Remarks	Description
Att_id	Integer	Non-null	The attribute id
Obj_id	Integer	Non-null	The object identifier
Att_value	Date	Non-null	The date value

MCS_TIME_ATTRIBUTES

Field Name	Type	Remarks	Description
Att_id	Integer	Non-null	The attribute id
Obj_id	Integer	Non-null	The object identifier
Att_value	Time	Non-null	The time value

MCS_DATETIME_ATTRIBUTES

Field Name	Type	Remarks	Description
Att_id	Integer	Non-null	The attribute identifier
Obj_id	Integer	Non-null	The object identifier
Att_value	Date/time	Non-null	The value of the identifier

5.8 Annotations

MCS provides annotation capability to users. Users can create multiple annotations for logical files, collections or views. MCS preserves the timestamp of the annotations so that users can query the MCS for annotation based on time and/or user who created the annotation. The following table captures the information about the annotations.

Field Name	Type	Remarks	Description
Obj_id	Integer	Non-null	The file or collection identifier to which the comments belong to
Annotation	Varchar(250)	Non-null	The annotation provided by the user
User_dn	Varchar(50)	Non-null	The user who provided the annotation
Date	Date	Non-null	The date/time can be auto generated

8/5/2002

Time	Time	Non-null	
Type	Integer	Non-null	The type of the object referred to 0 – logical files 1 – collections 2 - views

5.9 Creation history

The following table records the information about how data items are generated. Initially, the table provides only a textual description of the creation history and the date on which the data object was created. This information may be used to recreate the data item if it ever gets corrupted, or the application may decide to recreate the dataset if the cost of recreating it is less than the cost of retrieval of the data item. Eventually, we may add more attributes to allow more sophisticated queries, for example, on the analysis program run or the experimental apparatus or input conditions.

MCS_CREATION_LOG

Field Name	Type	Remarks	Description
Data_id	Integer	Non-null	The data identifier
Creation_desc	Varchar(250)		Description of how the data was created. This may be a record of the analysis program run along with input files and parameters, or it may be a record of experimental apparatus and conditions
Creation_date	Date/time		The timestamp of the creation of the data file (not the creation of the record in the metadata service)

5.10 Accessing external catalogs

Given the fact that many virtual organizations may have their own metadata catalogs, it becomes important to include information required to access these external catalogs. The user or application can then use this information to further query the external catalog. The following table provides this information.

MCS_EXTERNAL_CATALOGS

Field Name	Type	Remarks	Description
Data_id	Integer	Non null	The data id
External_Data_id	Integer	Non null	The domain or application specific data identifier used in the external database

8/5/2002

Database_type	Varchar(250)	The external database type i.e. relational or XML
Database_name	Varchar(250)	The name of the database
Database_desc	Varchar(250)	General description of database content
Table_name	Varchar(250)	The link table in the external database
Host_name	Varchar(250)	The hostname where the external database is located
Host_address	Varchar(250)	The ip address of the host where the database is located

6 Future Directions

This is our first iteration for the design of the Metadata Service. We invite feedback from application communities that will be used to refine the metadata schema presented in this paper.

A functionality that we envision for the near future for the Metadata Service is the ability to keep track of “invocations” in the context of GriPhyN’s virtual data system. An invocation is a particular transformation or a set of transformations applied to specific data items. The goal of GriPhyN (www.griphyn.org) is to provide transparency with respect to data location as well as data materialization. To support on-demand, automatic data creation, the exact way the data was produced needs to be recorded along with other metadata about the resulting data products. This metadata about the transformation (invocation metadata) needs to contain information such as which analysis was performed on the data, what where the inputs, and parameters used, environmental variables etc. As we understand these issues in greater detail, we will augment the current design to support this functionality.

7 References

- [1] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC Storage Resource Broker," presented at Proc. CASCON'98 Conference, 1998.
- [2] I. Foster, A. Iamnitchi, M. Ripaenu, A. Chervenak, E. Deelman, C. Kesselman, R. Schwartzkopf, L. Guy, W. Hoschek, P. Kunszt, H. Stockinger, K. Stockinger, and B. Tierney, "Giggle: A Framework for Constructing Scalable Replica Location Services.," SC 2002, to appear, 2002.
- [3] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," in *ACM Conference on Computers and Security*, 1998, pp. 83-91.

- [4] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "A Community Authorization Service for Group Collaboration," presented at IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.