

**Representing Virtual Data:
A Catalog Architecture for Location and Materialization Transparency
— Draft of January 26, 2001 —**

Ewa Deelman, Ian Foster, Carl Kesselman, Miron Livny

1	Introduction.....	2
2	Architecture Overview.....	2
3	A Simple Example.....	5
4	Catalog Details.....	6
4.1	Metadata Catalog.....	6
4.2	Replica Catalog.....	7
4.3	Derived Meta Data Catalog.....	8
4.4	Derived Data Catalog.....	8
4.5	Transformation Catalog.....	9
5	Detailed Example.....	10
	Acknowledgments.....	12
	Bibliography.....	12

1 Introduction

Data Grid systems can facilitate distributed access to, and analysis of, large amounts of data by providing:

1. Data access and replication services focused on enabling *transparency with respect to location* as a means of improving access performance (with respect to speed and/or reliability). These services will provide for secure, reliable, and high-speed access to metadata and data, as well as replication and distribution of data in wide area environments.
2. Data derivation services focused on enabling *transparency with respect to materialization*, as a means of facilitating the definition, sharing, and use of data derivation mechanisms.

These two transparencies are sometimes referred to collectively as “virtual data.”

We have argued elsewhere [5] for a clean separation between (1) the data access, replication, and derivation services used to operate on virtual data and (2) the catalogs used to characterize virtual data with respect to its properties, where it is located, and (for derived data) how it is generated.

The design of such catalogs is basically a database design problem. While we do not yet know enough about virtual data to be able to say anything definitive about this problem, we can (and do in this article) propose a candidate architecture to be explored in future research.

2 Architecture Overview

Our candidate catalog architecture features five distinct catalogs.

- The metadata catalog (MDC) maps from some set of attributes to logical file names. This element is a conventional metadata catalog [3], with the exception that it maps from attributes to logical rather than physical file names.
- The replica catalog (RC) maps logical files to one or more physical files, hence providing the information required to obtain access to a physical copy of a logical file. This element provides the information required to support data replication in a Data Grid environment [1, 4].
- The derived data catalog (DDC) records information associated with derived datasets, such as the transformation needed to generate the derived dataset, the cost of that transformation, and a unique derived dataset name. “Meta attributes” of derived data, such as owner, name, author, and creation-time are also recorded here. This element, and the two that follow, introduce the additional information required to support the manipulation of derived data.
- The derived metadata catalog (DMDC) is similar to the MDC but it maps from a set of application-specific attributes (these attributes are specific to the physics experiments producing the data, such as the instrument channel name in LIGO) to a derived data id(s) that indexes into the DDC.
- The transformation catalog (TC) records information about the transformations used to create derived data: executable name, arguments, etc.

These five elements operate as follows (see).

- Each materialized dataset (whether “raw” data, or the result of a derivation) is represented by an entry in the RC. One or more MDC entries will typically also be provided, to support access by attribute rather than name. An MDC query returns the logical file name(s) of materialized datasets that match the supplied attributes. A query to the RC will then return the physical file name(s) associated with a supplied logical file name. The RC and MDC thus provide *transparency with respect to location*.
- Each derivation procedure registered with the Data Grid is represented by an entry in the DDC. This entry includes a unique derived data id plus a transformation skeleton (e.g., F(P).X) comprising a transformation name (F, in our example), the names of any logical files to which the derivation program is to be applied (X), and information about any parameters (P). One or more DMDC entries are typically also provided for the derivation procedure, to support access to DDC entries by attribute rather than id. An DMDC query returns one or more derived data ids, corresponding to the way(s) to materialize the requested data. A DDC query returns the transformation skeleton that matches the supplied id. The TC then provides information about the program that must be executed to perform a derivation, such as its name, location, and cost. The DDC, DMDC, and TC thus provide *transparency with respect to materialization*.
- The execution of a derivation procedure generates a new materialized dataset. A new entry is then created in the RC (and MDC) so that other programs can discover and access it.

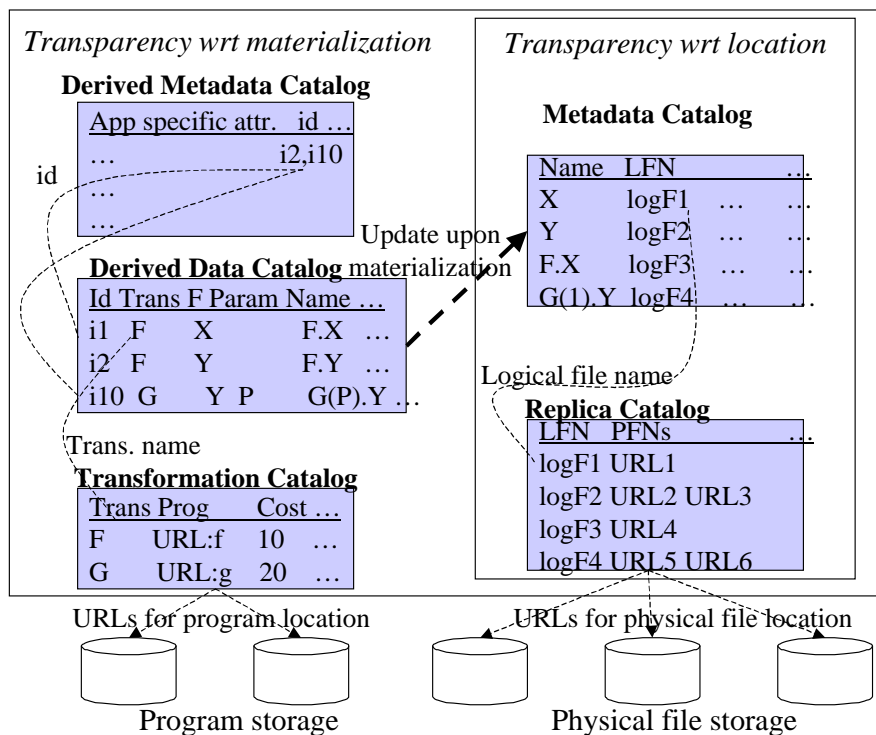


Figure 1: Virtual data catalog architecture, showing the five catalogs with some of their attributes and (as dashed lines), the linkages between them. The example derivations and file names are used to illustrate discussion in the text.

An issue to be determined is the nature of the reference from the DDC to the TC. We propose here that the reference be simply the transformation name. But some TC attributes such as cost may depend on other DDC values: for example, $\text{cost}(F.X)$ may be different than $\text{cost}(F.Y)$.

This is clearly not the only possible catalog organization. One alternative would be to eliminate the DMDC and maintain all information about data, whether materialized or not, in the MDC. We could then use standard query mechanisms to manipulate materialized and unmaterialized data, distinguishing between the two only when it came time to access data. (The RC would be extended with DDC information, so that a RC lookup would reveal whether or not a requested data item was materialized.) However, we are concerned that achieving this degree of transparency will be difficult when we know so little about how to represent derived data. For example, it is easy to define derived data sets of infinite extent. How do we represent such datasets in an MDC? Maintaining the DDC and DMDC separately from the MDC and RC simplifies experimentation with derived data representations. It also makes it easier to use standard MDCs and RCs, which is important because many application domains already have MDCs.

Another possibility is to merge the DMDC and DDC. However, this introduces application-specific attributes into the DDC structure, making it less generic. There are several advantages to having separate catalogs:

- We separate logically distinct components: the DMDC helps us to identify the data requested and the DDC tells us how to produce that data.
- We recognize that there may be several ways to obtain a desired data, so that maintaining separate catalogs is more efficient. For example, in LIGO, if we request a channel C in the time interval 100-110, this piece of data can be constructed in more than one way. For example we can concatenate smaller time intervals of channel C or extract the 100-110 interval from a longer time interval of C. This results in one entry in the DMDC containing two derived data ids, which match two entries in the DDC, one representing concatenation and the other extraction. (See Section 5.)
- Constructing a generic DDC simplifies implementation issues, as application-specific attributes need not be considered in the queries.

The process of data access is depicted in Figure 2. First, the attributes used to identify the data in the physics domain are used to query the MDC.

If the data has been materialized (either through data derivation of as raw data), the MDC will provide the logical file name in which the data is present. This name is then used to find the corresponding physical file names and their locations (via a query to the RC). Finally, this information is used to access the data.

If the data has not been materialized, the attributes are used to find the transformation(s) that can produce the data (by querying the DDC). We assume that the query is well constructed and that a means of materializing the data is provided. The DDC provides the logical input file names and transformation parameters, which are needed to materialize the data. Additionally, the DDC provides the necessary transformation's name. Using that name the TC provides the code used in the execution of the request. In order to materialize the data, the RC might need to be consulted to provide the physical input file name and their locations. (Of course, the input file itself might not be materialized, which would require a new chain of DDC and TC queries (not shown)).

Once the data is materialized, it needs to be published in the MDC and RC. In order to achieve this, additional data processing might need to occur. Information about how to publish the data is specified in the DDC along with the logical file name.

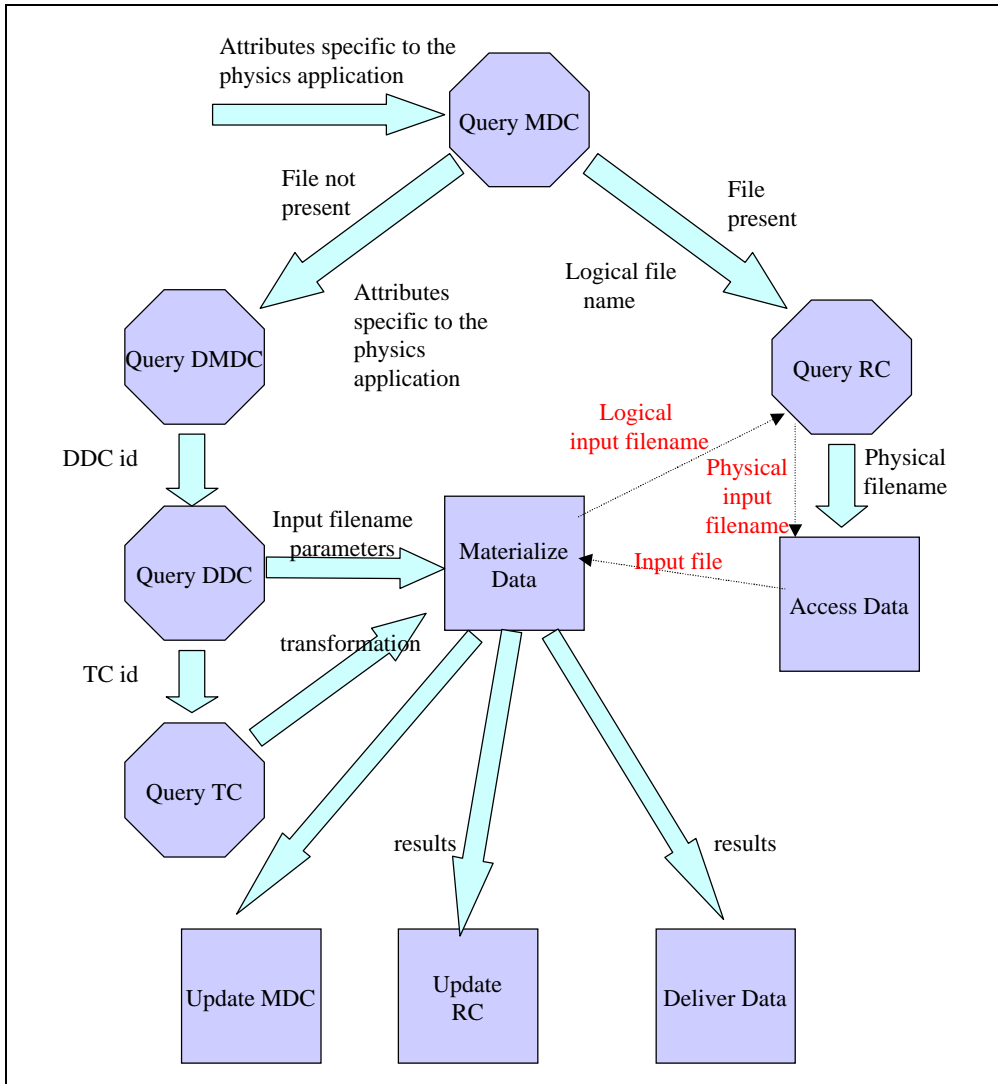


Figure 2: Simplified Process of Data Access.

3 A Simple Example

We use a simple example to illustrate how we see these catalogs operating. (A more complex example in the context of LIGO can be found in a later section.) We consider a series of cases of increasing sophistication.

Materialized data and known file name. We first consider the case in which the data is already materialized and we know the logical file name for the desired dataset. In this case, we need simply to consult the RC to determine the location of the corresponding physical file(s). For example, if the logical file name is logF2 (see figure) then we will determine that there are

physical copies of this file at locations URL2 and URL3. In order to select between these two copies, we might then consult the MDC to determine the size of logF2 and an information service to determine transfer costs and resource availability.

If we did not know the logical file name, we would of course consult the MDC (supplying appropriate attributes) to determine the name to be supplied to the RC.

Unmaterialized data, parameter-less derivation program. We next consider a simple case in which the required data has not been materialized. Let us assume that the unique id of the data of interest is known, and let that id be i_1 . Querying the DDC with id i_1 , we find out that the desired data can be produced by running a program F on a dataset X and will result in a data product named $F.X$. As illustrated in , the DDC entry records that to produce $F.X$, we need to invoke a transformation F on file X , with in this case no other parameters. (The DDC also records that to produce $F.Y$, we run F on file Y .) Consulting the TC then reveals that transformation F corresponds to the program at location $URL:f$ and also provides other information about the program, such as its cost. (The executable location might itself be a logical rather than a physical name, but we do not consider that issue here.) This information allows a request planner to determine how to generate the required data. When request execution is complete, we add an entry to the MDC (“ $F.X$ logF1”) and RC (“logF1 URL1”) to indicate the availability of the new physical data. Initially, we assumed that the id of the derived data was known, however, if this is not the case, the id can be obtained by querying the DMDC.

Derivation program with parameters . The programs used to produce derived data may have parameters that can take different values. It will often be impractical to create a DDC entry for every possible parameter value. Instead, the DDC can specify that a transformation program has parameters and indicate how the name of the materialized data produced by running that program is to be created. For example, in the DDC includes an entry for a program G that takes a single parameter. This entry tells us that running G with parameter values $P=1$, $P=2$, etc., on file Y produces materialized data $G(1).Y$, $G(2).Y$, etc.

4 Catalog Details

In general, we view catalogs as being organized and maintained at the community level. For example, a given community will decide how to organize the materialized data, will develop a set of commonly agreed upon transformations and make them available. Once the information is entered in the catalog, it cannot be removed or modified. We place this restriction, because of the interdependencies between the catalogs. For example, the MDC has links to the DDC to indicate how the data was materialized (if the data was not raw). If then the DDC entry is allowed to be modified, the consistency will be broken. The same argument applies to the TC.

We now provide a detailed description of the attributes required in each of the catalogs.

4.1 Metadata Catalog

The MDC records attributes that can be used to identify a particular data item, such as the following:

- Physics application-specific attributes, such as channel name, time interval boundaries, etc...
- File size
- Creator’s name and modifier’s name.
- Creator’s timestamp and modify timestamp.

- Logical file name: that is, a reference to an RC entry.

It is frequently the case that applications that are to use Data Grid services already have MDCs. (For example, LIGO maintains information about events and their associated data in a DB2 database.) Such application-specific MDCs require just two relatively small changes to be used within a Data Grid environment. First, the physical file names typically provided in an MDC must be replaced with a logical file name: that is, with a reference to the RC entry for the file. Second, if we are dealing with derived data, then we also need to maintain information about how the data was materialized.

MDC attributes about derived data are intended to provide sufficient information to regenerate the derived data. They include, firstly, an attribute naming the appropriate DDC entry, which of course names the transformation used to generate the data, the input files and parameters passed to that transformation program, and so forth. Hence, the MDC must contain attributes:

- DCC entry id: a foreign key to the DCC entry that describes how the data was created.
- The parameter values passed to the transformation program.

In addition, we might want to provide the following attributes:

- Information about the computer system on which the transformation was run, if this affects the result obtained.
- Execution's runtime, how long it took to materialize data.

A complicating factor that we need to explore further is that a materialized dataset might be generated via a series of transformations (e.g., the example of Section 5), in which case the MDC entry presumably needs to contain multiple DDC entries.

4.2 Replica Catalog

We take the RC structure defined in [1] as a starting point. A complicating factor in this design is that files are grouped into collections so as to facilitate management and manipulation of related files; files can be manipulated both individually and at the collection level. Our DDC, DMDC, and MDC designs should presumably also support collections. However, we do not address that issue here.

RC collections are user-defined and allow users to group together and manipulate files that are logically related. Collections contain the following (optional) attributes:

- File name, creator's name, modifier's name (most recent modifier) and corresponding create time timestamp and modified time timestamp.

The mapping of a particular file collection to a physical location is done by "locations." Each location entry corresponds to exactly one physical storage system. Associated with locations are one required attribute:

- URL constructor,

and nine optional attributes:

- Filename: list of the filenames at the location.
- Hostname: host where the location is present.
- Protocol used to access the files.

- Port required to access the system.
- Path where the replica files are located.
- Creator's name and modifier's name of the location entry.
- Creator's timestamp and modify timestamp of the location entry.

In addition, we note that decision procedures used to perform proactive data replication may require information about how often given data is accessed and when it was accessed last. This information can be maintained in the RC. An issue to be addressed in the future is whether this information is best kept on a per-file, per-collection, or per-location basis. Maintaining information on a per-file basis can allow for fine-tuning of replication decisions, but may introduce significant overhead. If however, the information is kept on a per-location basis, the replica creation decisions are coarse, such as replicate the entire location, regardless of which files are frequently accessed. A point in favor for per-location performance is that even if only one file in the location is often accessed, it can affect the access times of other less-used files, so replicating the entire location might make sense.

If we decide to maintain performance information on a per-location basis, we need to add two new optional attributes to the location object, such as the number of times a location was accessed and when was the last time the location was accessed.

To maintain per-file performance information, we can use the logical file structure present in the replica catalog. We can use this structure to maintain usage information of the files such as how many times the file was accessed, when was the last time the file was accessed.

Additional information needed for proactive replica creation, such as average storage system bandwidth, can be obtained from the information services.

4.3 Derived Meta Data Catalog

DMDC information helps identify the desired derivation procedures. The DMDC implements a (possibly one-to-many) mapping of application-specific attributes to the DDC, which describes how to produce the desired data.

The following attributes are required:

- Application-specific attributes, such as event time in CMS.
- Derived data id, which indexes into the DDC.
- Creator's name and modifier's name.
- Creator's timestamp and modify timestamp.

4.4 Derived Data Catalog

The DDC contains information necessary to produce a specified (via an id) data item.

The following are required attributes:

- Derived data id (we need to be able to generate unique ids).
- Transformation id, used to identify the program used to produce the data. This id indexes into the TC.
- Input file(s) name(s), names of the files needed by the program, these can be raw data files, or derived data ids, if the input file is a result of a transformation.

- Parameter list, list of parameters used by the transformation.
- Name, resulting materialized data name. This name indexes into the RC.
- Creator's name.
- Creator's timestamp.
- Publish procedure: instructions on how to place (and possibly further process) the data into the MDC and RC.
- Valid field: since we provide only write once capability for this catalog, the entry is used to invalidate erroneous derived data descriptions.

4.5 Transformation Catalog

The transformation catalog maintains all the necessary information about the specific transformations used to produce derived data.

The following attributes are required:

- Transformation id.
- Program name.
- The name of the creator.
- The time of creation.
- Version number.
- Supported architecture and OS: the architecture and OS for which the program has been compiled.
- URL: repository(s) where program may be found, and path in the repository.

In addition, we can specify the following optional attributes

- Language the program is written in.
- Configuration file (name and location) containing the parameters for the program and their default values.
- The compiler used to generate the program.
- Transformation description.
- Cost of running program.
- Logical expression specifying the resource requirements, such as min of processors, maximum number of processors (by default we can assume a single processor system), use only a square number of processors etc...
- Valid field: since we provide only write once capability for this catalog, the entry is used to invalidate erroneous or transformations.

5 Detailed Example

We use an application-specific example to illustrate how we see the various Virtual Data catalogs operating. We consider LIGO data, which comprises a number of “derived” channels computed from physical data via the application of various transformations [2]. LIGO data is stored in a “frame file,” where each file contains exactly two seconds of data from a collection of channels. One of these channels is the actual instrument data, while the remaining channels contain engineering data that is used to condition the instrument data to determine if a significant event occurred. We consider a series of cases of increasing sophistication.

Materialized data and known file name. We first consider the case in which the data is already materialized and we know the name of the file in which the data was stored. In this case, we need simply to consult the replica catalog that contains this information. For, example, if we want time 100-110 from channel C, the file might be named, C-100-110, and a replica catalog will tell us where to find a copy of that file:

RC	Logical File Name	Locations
	C-100-110	ftp://hpss.sdsc.edu/ligo/C-100-110, ftp://ligo.caltech.edu/cache/C-100-110

If we did not know the name of the file, we would of course consult the MDC to determine the name.

To support the ability to derive performance estimates, we need to acquire information relating to size of data, and transfer costs. Size information is present in the MDC, and the transfer costs can be obtained using the information service.

Unmaterialized data, parameter-less derivation program. We next consider a simple case in which the data has not been materialized. (In order to determine whether a given data item is materialized, we can query the RC with the logical file name for the data, or if this name is not available, we can query the MDC using application-specific attributes. If the data is not present either of the above queries will indicate that fact). We assume that the data for channel C is contained in a file named LA-LIGO containing multiple channels from the time interval 100-110. We assume also that we can extract Channel C from this file into a separate frame file, hence materializing the dataset C-100-110, by running a program *extract-c*. So to materialize our virtual data, we need to know that if we are looking for data file C-100-110, we have to run *extract-c* to generate the data. We keep this information in the derived data catalog:

DDC	Id	Transformation	Input File	Derived Data Name
	i10	extract-c(LIGO-100-110)	LIGO-100-110	C-100-110

Given the above derived data catalog, a request planner can determine how to generate the required data. When the request execution is complete, we add an entry to the replica catalog to indicate the availability of the new physical data.

In some cases, it might be important to keep track of exactly how *extract-c* was run, including what version of the code was used, what compiler was used to generate the program, who wrote the program, what machine it ran on, etc. We consider this auxiliary information to be metadata associated with the materialized derived data. As such, it is no different than any of the other metadata associated with this file and is associated with the metadata catalog deployed as part of the overall data management solution.

Derivation program with parameters . There are several problems with the catalog structures described above. We would like to have the ability to have a general extract program that we can use to get any channel. We could achieve this by generalizing *extract-c* to become *extract-*

channel(arg) where *arg* is the name of the channel to be extracted. We could therefore construct a DDC that looks like:

DDC	Id	Transformation	Input File	Derived Data Name
	i10	extract-channel(C,LIGO-100-110)	LIGO-100-110	C-100-110
	i25	extract-channel(D,LIGO-100-110)	LIGO-100-110	D-100-110

This is still limited by the fact that we must enumerate all of the potential derived data files in the derived data catalog. In the case of channel names, this is possible, although somewhat inconvenient. A more flexible solution might be to have a single entry to represent all the channel extractions

DDC	Id	Transformation	Input File	Derived Data Name
	i40	extract-channel(%channel)	LIGO-%(start)-%(stop)	%(channel)-%(start)-%(stop)

We can take this one step further, by considering the fact that LIGO frame files are generated in two second intervals, thus we need to concatenate 5 files in order to get the required times.

DDC	id	Transformation	Input File	Derived Data Name
	i40	extract-channel(%channel)	i22	%(channel)-%(start)-%(stop)
	i22	concatenate(%F)	%F= { LIGO-%(t0)-%(t1) t0 ≥ start, t1 ≤ stop }	LIGO-%(start)-%(stop)

Thus using this DDC, a request planner can create a series of program executions that will concatenate the appropriate frame files and extract a channel. Note that we have defined the concatenation program to work only over two second intervals. Thus if one wanted a data set starting at an odd second, the end application will still have to manipulate the returned data to find the appropriate starting point. In order to promote reuse, we might want to make further restrictions, such as limit the derived data to start on a day or week boundary, etc. Thus we conclude that virtual data does not remove all data manipulation responsibilities from the application.

Note: The input file for the first entry is an id indexing into the DDC. It is possible that this file was already materialized, so the derived data name for the entry i22 needs to be instantiated and the existence of the file established by querying the RC.

The template language proposed in the preceding example is quite limited and will clearly prohibit many interesting types of derived data. An interesting research question will be to determine what type of language should be used to express derived data.

In the beginning of our discussion, we assumed that we knew the name of the file we were looking for. In general, this may not be the case. In such situations, we would rely on a meta-data catalog to map the properties of interest into the appropriate file names. For example, the following MDC might be defined for physical data: C-100-110:

MDC	StartTime	StopTime	Channel	Instrument	File Name
	100	110	C	LA	C-100-110

Likewise, the DMDC would map attributes into virtual data file names.

DMDC	StartTime	StopTime	Channel	Derived Data Id
	%start	%stop	%channel	i40, i57

Note: It is possible that a desired data can be constructed in more than one way. For example we can concatenate smaller time intervals of channel C or extract the 100-110 interval from a longer time interval of C.

The final catalog defined in our architecture, the *transformation catalog*, is used to map program names into specific executables. For example:

TC	Program	Executable Name	Version	Repository	Cost
	extract-channel	extract.exe	1.0b	cvs://code.griphyn.org	O(fileSize)

The schema described here is very similar to what has evolved over the past 2 years within LIGO.

Acknowledgments

We are grateful to our colleagues within the European Data Grid, GriPhyN, and Particle Physics Data Grid projects for numerous helpful discussions on the topics presented here.

Bibliography

1. Allcock, B., Bester, J., Bresnahan, J., Chervenak, A.L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D. and Tuecke, S., Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. In *Mass Storage Conference*, 2001.
2. Allen, B., Deelman, E., Kesselman, C., Lazzarini, A., Prince, T., Romano, J. and Williams, R. LIGO's Virtual Data Requirements, California Institute of Technology T000135-00-D, 2000.
3. Baru, C., Moore, R., Rajasekar, A. and Wan, M., The SDSC Storage Resource Broker. In *Proc. CASCON'98 Conference*, 1998.
4. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets. *J. Network and Computer Applications*, 2001.
5. Foster, I. and Kesselman, C. A Data Grid Reference Architecture, 2001. In preparation.