

# Data Location Interface for the Workload Management System

Draft v0.4

Heinz Stockinger, Flavia Donno

September 2, 2004

## Abstract

The Matchmaker of the Workload Management Systems (WMS) takes care of finding a suitable computing resource (i.e. Computing Element) to execute a data intensive job. The *location* of the required *input data* is one of the most important input parameters for the matchmaking process. Therefore, the Matchmaker requires a uniform *query interface* to locate data stored in Storage Elements. The following document describes the basic **Data Location Interface** as well as some required changes to the Workload Management System.

## 1 Introduction

In the current version of the Workload Management System in LCG 2.x [1], the Matchmaker uses file locations as the base for determining where input data is located. Currently, the Replica Location Service (RLS) is queried via the replica manager method `listReplicas`:

```
std::vector<std::string> listReplica(const std::string& lfn);
```

Based on feedback from several application users, the following new requirements need to be addressed:

- **Datasets rather than logical files:** Many application users have expressed the requirement that they want to deal with sets of files (collections or datasets) rather than single files addressed by Logical File Names. Therefore, the concept of a **Logical Data Set (LDS)** is required. In addition, it is assumed that experiment specific dataset catalogues are provided by VOs (e.g. physics experiments). Such catalogues describe dataset characteristics and allow for locating of replicas of datasets.
- **Allow for a generic query string:** Once a dataset catalogue is in place, application users typically express their input data requirements in an experiment specific way to query data sets. The query is then resolved by the dataset catalogue which returns locations of the requested datasets.

- **Unique query interface to data catalogues:** The RLS provided by the EDG project is a candidate file replica catalogue. However, there are currently several other ongoing projects that are building replica catalogue systems. In order to shield the WMS from several different implementations, a unique, standardised query interface to such catalogues is required.

In order to satisfy these requirements, we consider both types of catalogues (dataset catalogues and file replica catalogues) as *data catalogues* which need to provide

- the same *standard interface*,
- the same agreed *protocol*

in order to be used by the Matchmaker. We refer to this as the **Data Location Interface** and give specific details in Section 2. The consequences for the Workload Management System and possible extensions to the JDL (Job Description Language) are then discussed in Section 3.

## 2 Data Location Interface

The Matchmaker only requires a basic query (search) interface in order to obtain the location of required input data. Therefore, the proposed interface is limited to very specialised queries. The interface does not include general catalogue commands such as insert and delete operations.

In general, the Data Location Interface needs to allow for the following types of `InputData` variables:

- File - Logical File Name (LFN)
- Global Unique Identifier (GUID)
- Dataset - Logical DataSet (LDS)
- Query - Generic query

In order to distinguish the four different data types, the `listReplicas` method requires the `InputData` type to be stated explicitly. The basic query interface is as follows:

```
std::vector<std::string> listReplicas(std::string inputDataType,
                                     std::string inputData)
```

**Input Parameters:** The parameter `inputDataType` corresponds to one of the four `InputData` types listed above. It can have one of the four values: `lfn`, `guid`, `lds` or `query`.

The parameter `inputData` corresponds to the value of the data type indicated by `inputDataType`. `inputData` should be well formed but it is up to the catalogue implementation to do necessary syntax checks. In this way, we allow for a most flexible, general interface.

**Return value:** The method returns a list of URLs referring to the data location. Each URL includes a Storage Element (DNS hostname) the corresponding access protocol. If no URL is found for the given InputData, NULL is returned and a NoURLFound exception is thrown. For datasets, the URL of a single file in the dataset is sufficient, assuming that all other files within the dataset are located at the same Storage Element and accessible via the same protocol.

**Exceptions/SOAP Faults:**

- **InvalidInputDataTypeException** - in case a catalogue does not support the indicated InputData type. For example, a file replica catalogue does not need to support the data type “lds”.
- **InputDataException** - in case the specified InputData does not exist.
- **NoURLFoundException** - in case for the given InputData, no URL is found.
- **CatalogException** - in case of a generic catalogue error.

**Example of query output:**

```
srm://example.org/data/file1  
http://example.org/directory/dataset
```

Note that “example.org” corresponds to a valid SEid (i.e. the DNS hostname of a valid Storage Element) that is registered with the information system. Therefore, all returned URLs *must* follow these conventions.

An alternative interface is to request a set of input data items and then return InputData/URL pairs similar as defined in [2]:

```
stringPair<std::string, std::string> listReplicas(std::string inputDataType,  
                                                vector<std::string> inputData)
```

A generic data type for stringPair still needs to be defined.

**Note**

Although we propose the same interface for both file replica and dataset catalogues, the following assumptions are valid:

- A replica catalogue (providing LFN-PFN mapping) does *not* need to implement LDS based queries nor need to understand the concept of LDS.
- A dataset catalogue (providing a LDS-Location mapping) does *not* need to implement LFN(GUID) based queries nor need to understand the concept of LFN(GUID). For example, if the InputData type `lfn` is used for the dataset catalogue, the catalogue is free to return an `InvalidInputDataTypeException`.

In this way, different catalogue types can be used. Each catalogue is free to implement a subset of the given data types. In addition, further data types might be added in the future if necessary.

## 2.1 Protocol

Since **SOAP** is currently one of the most commonly used protocols in the international Grid community, we propose to use SOAP as the request-response protocol between the Matchmaker (client) and a Data Catalogue.

A WSDL description of the interface described above will be provided in the next version of this document.

We are aware that SOAP has a performance overhead with respect to a customized wire protocol but if performance becomes a bottleneck, a “more efficient” protocol might be chosen in the future.

## 2.2 Security

A secure connection between the requesting client and the Data Catalogue is required. We assume that a standard security interface provided by EGEE (or another project) can be used for that purpose.

## 2.3 Optimisation Interface

The original replica management interface available in the latest EDG release 2.1 provided the following method for optimised data access:

```
getAccessCost(const std::vector<std::string>& lfns,  
              const std::vector<std::string>& ces,  
              const std::vector<std::string>& protocols)
```

Since in LCG-2 the Replica Optimization Service is not used, the method does not return the correct values. Therefore, we currently assume that the following method does not need to be implemented by the Data Catalogue Interface unless an optimisation service is in place.

## 2.4 Dataset Issues

In order to make efficient use of a dataset (collection), all files for a given dataset need to be in a single location, i.e a single Storage Element. For now we assume that whenever a dataset catalogue returns the URL of a dataset, *all* files need to be located at the given URL (i.e. in the Storage Element referred to by the URL). This is an important limitation but we consider this as the best starting point.

In the future, more enhanced issues such as dealing with partial datasets need to be addressed.

The dataset catalogue should also have be able to use some kind of validation mechanism to determine if datasets are fully available at a Storage Element.

This is an important issue since the Matchmaker bases its scheduling decision on this assumption.

### 3 Workload Management System Changes

In order to allow for a generic Data Location Interface, some changes in the Workload Management System are required. Since the Matchmaker is now extended to contact a generic Data Catalogue, an additional JDL field is required that indicates the URL of a “non-standard”, application specific Data Catalogue.

For backward compatibility, the default catalogue for a particular VO is obtained from the information services. For example, one VO might decide to use RLS as the default catalogue whereas another one might chose to use a dataset catalogue. If a different, non-default catalogue is used by specifying the `DataCatalog` variable, the Matchmaker uses that catalogue rather than the default.

In addition, the four `InputData` types LFN, GUID, LDS and Query need to be supported. Each of them has a the respective *prefix* as indicated in the

- `lfn:validLfnString`
- `guid:validGuidString`
- `lds:validDatasetString`
- `query:validQueryString`

JDL example for using an experiment specific dataset catalogue with an LDS as `InputData`:

```
[
DataCatalog = {"http://example.org/CMSDataSetCatalog"};
InputData    = {"lds:mu03_tt_4mu/mu_Hit245_2_g133"};
]
```

In the example above, the CMS physicist specifies that she needs a certain dataset as `InputData`. The catalogue specified by `DataCatalog` is the CMS dataset catalogue that provides the interface described in Section 2.

#### Internal Details

Although the application user has to indicate the requested `InputData` type by using the a prefix, the Workload Management System internally calls the respective Data Location Interface method.

We foresee to provide a plug-in interface for the Matchmaker that allows for the following two options:

- Call the new Data Location Interface that in turn contacts a data catalogue via SOAP.
- Use the current interface to RLS for backward compatibility.

## 4 Open Issues

For some catalogue implementations it seems to be more efficient to just return a list of Storage Element hostnames than entire URLs of the specified InputData. An additional method `listReplicaHosts` could be used in such a case:

```
std::vector<std::string> listReplicaHosts(std::string inputDataType,
                                         std::string inputData)
```

For the matchmaking decision, it is enough to obtain the Storage Element hostnames, but once the specific job has arrived on the Worker Node of a Computing Element (close to the selected Storage Element), more detailed information like the full URLs are often required. We therefore reconsider this method once we have gained experience with matchmaking and datasets.

In general, the **job execution environment based on the dataset matchmaking results**, needs to be further analysed.

## 5 Conclusion

The proposed Data Location Interface is the essential for the Matchmaker to contact a Data Catalogue. We propose that upcoming file, replica or dataset catalogues provide this interface in order to be used effectively by the Matchmaker.

The proposed interface will be implemented in the short term for the LCG-2 Matchmaker. We expect a first prototype in autumn 2004. A more long term solution is to apply the DataLocationInterface or a variation also to the Matchmaker in EGEE.

## References

- [1] Workload Management System (WMS) in EDG and LCG-2:  
<http://server11.infn.it/workload-grid/>
- [2] EGEE Design Team, EGEE Middleware Design - Draft, EGEE\_DJRA1.2-487871.-v0.4, 18 August 2004.