*HPC Data management prototype*

Design DOCUMENT

Version *<1.0>*

*1/28/2016*

**Version History**

[Provide information on how the development and distribution of the Design Specification will be controlled and tracked. Use the table below to provide the version number, the author implementing the version, the date of the version, the name of the person approving the version, the date that particular version was approved, and a brief description of the reason for creating the revised version.]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version Number** | **Implemented**  **By** | **Revision**  **Date** | **Approved**  **By** | **Approval**  **Date** | **Description of Change** |
| 1.0 | *Prasad Konka* | *1/28/2016* |  |  | *Initial Draft* |
| 1.0 | Prasad Konka | 2/3/106 |  |  | Changes based on Zhengwu Lu comments |
|  |  |  |  |  |  |

TABLE OF CONTENTS

[1 Introduction 4](#_Toc485476715)

[1.1 Purpose of The Design Specification Document 4](#_Toc485476716)

[2 General Overview and Design Guidelines/Approach 4](#_Toc485476717)

[2.1 Assumptions / Constraints / Standards 4](#_Toc485476718)

[3 Architecture Design 5](#_Toc485476719)

[3.1 Introduction 5](#_Toc485476720)

[3.2 HPC DME 6](#_Toc485476721)

[3.2.1 Metadata 7](#_Toc485476722)

[3.2.2 Data Archival 12](#_Toc485476723)

[3.2.3 Data Discovery 15](#_Toc485476724)

[3.3 Data Sharing 15](#_Toc485476725)

[3.4 Audit 16](#_Toc485476726)

[3.5 policy management 16](#_Toc485476727)

[3.6 reports 16](#_Toc485476728)

[3.7 Hardware Architecture 16](#_Toc485476729)

[3.8 Software Architecture 16](#_Toc485476730)

[3.8.1 API Server 17](#_Toc485476731)

[3.9 Security Architecture 20](#_Toc485476732)

[3.10 Communication Architecture 22](#_Toc485476733)

[3.11 Performance 23](#_Toc485476734)

[4 System Design 23](#_Toc485476735)

[4.1 Database Design 23](#_Toc485476736)

[4.2 Application Program Interfaces 25](#_Toc485476737)

[4.3 System Integration 25](#_Toc485476738)

[4.3.1 Integrated Systems 26](#_Toc485476739)

[4.3.2 Code Organization 26](#_Toc485476740)

[4.3.3 IRODS Integration 27](#_Toc485476741)

[4.3.4 Globus Integration 30](#_Toc485476742)

[4.3.5 Cleversafe Integration 30](#_Toc485476743)

[4.4 User Interface Design 30](#_Toc485476744)

[4.5 System Performance 30](#_Toc485476745)

[Appendix A: Design Specification Approval 31](#_Toc485476746)

[APPENDIX B: REFERENCES 32](#_Toc485476747)

[APPENDIX C: KEY TERMS 33](#_Toc485476748)

# Introduction

## Purpose of The Design Specification Document

The Design Specification document documents and tracks the necessary information required to effectively define architecture and system design in order to give the development team guidance on architecture of the system to be developed. Design documents are incrementally and iteratively produced during the system development life cycle, based on the particular circumstances of the IT project and the system development methodology used for developing the system. These documents are initially created during the Planning Phase of the project and updated as necessary throughout the design process. The preliminary versions of these documents are reviewed during the EPLC Stage Gate Preliminary Design Review; the final versions are input to the EPLC project Detailed Design Review. Its intended audience is the project manager, project team, and development team. Some portions of this document such as the user interface (UI) may on occasion be shared with the client/user, and other stakeholder whose input/approval into the UI is needed.

The purpose of this design document is to define architecture and system design to develop a prototype high-performance computing data management environment that will provide a high-reliability storage model for underlying data objects including a dataset registration system, and an API for transfer of large datasets with no-loss of data. The dataset registration system will associate a label with a given managed dataset and will capture extensible metadata including security and access requirements for the managed dataset. This system will also be flexible enough to support export of metadata to a future system, development of service APIs to support integration with secondary systems.

# General Overview and Design Guidelines/Approach

This section describes the principles and strategies to be used as guidelines for designing and implementing the system.

## Assumptions / Constraints / Standards

The proposed design approach considers following SOA guidelines and standards at minimum.  These design principles create technology independent services and hence provide interoperability in the long term.

**Well Defined Services:**

Service interactions must be well-defined with a widely supported standard (e.g., RAML, WSDL) describing details to assist the service requestor to invoke the service(s) required.

**Loosely coupled Services:**

The service requester does not need to have any knowledge about the technical part of the service implementation. Therefore, business or technical in the background can be migrated or replaced by other technology, without affecting the service and therefore the service requester.

**Service Granularity:**

Services must be designed for appropriate granularities that offer greater flexibility to service requestors without impacting the performance and security. Services granularity should make it easy for service requestors to assemble services to execute business scenarios.

**Stateless Services:**

Services invocation must be independent of the state of other services and each service invocation has all the required information from one request to another.

**Discoverable Services:**

Services must be discoverable to support reusability of the services. Discoverability is also required to avoid development of solution logic that is already contained in an existing service.

**Use of Enterprise Integration Patterns:**

In order to promote loose coupling and composability, the design should consider following integration patterns.

**Enterprise Service Bus:**

An Enterprise Service Bus (ESB) represents an environment designed to foster sophisticated interconnectivity between services. It establishes an intermediate layer of processing that can help to overcome common problems associated with reliability, scalability, and communications disparity.

# Architecture Design

## Introduction

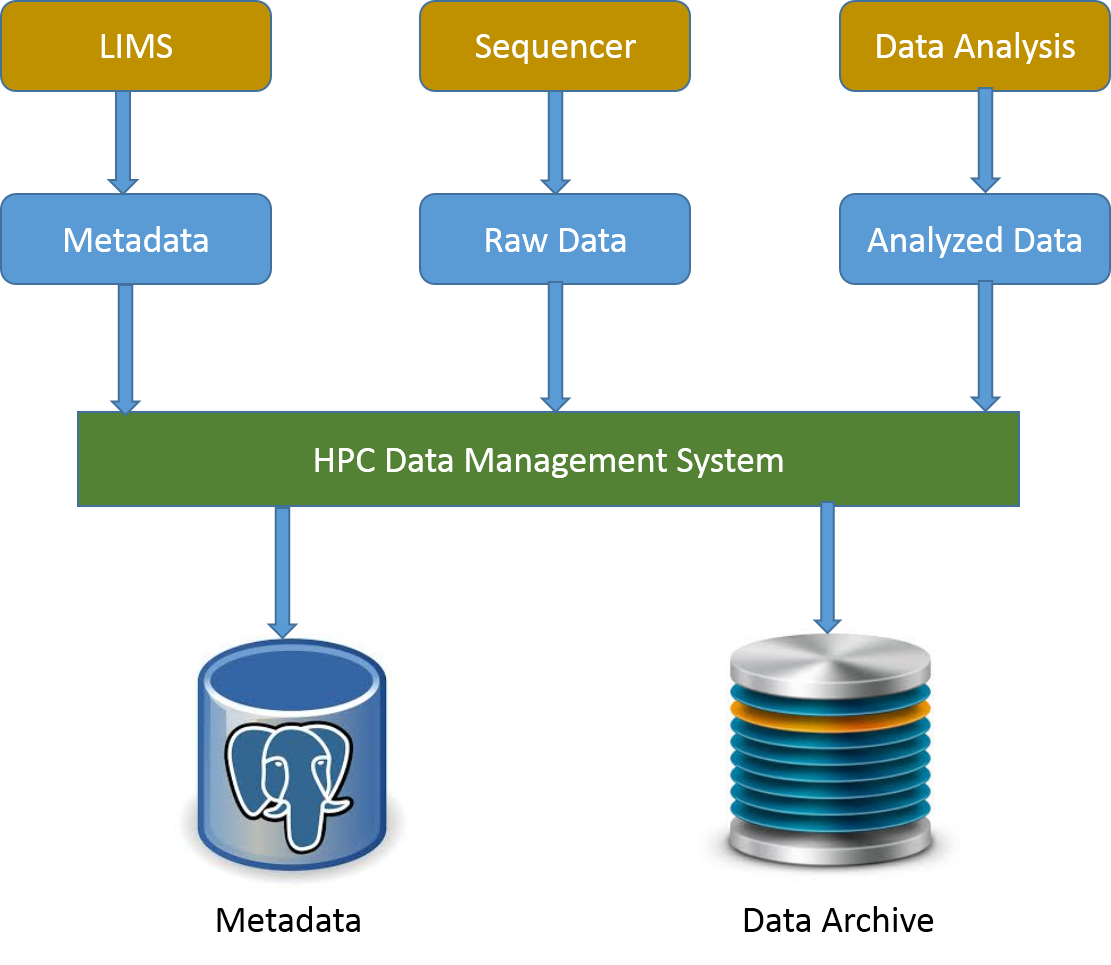
One of the most significant challenges to overcome for an effective high performance computing (HPC) support effort is effective data management. Data management is defined as effective tracking, annotation and storing of digital datasets, whether created by instrument or by software, accompanied with a data life cycle plan for these datasets. Without a reliable managed dataset solution, large datasets are frequently maintained in multiple copies across the physical storage in an isolated fashion, leading to an unnecessary expense as additional storage is required for analysis and storage of new data. A managed, secured, and high-availability solution will minimize the need for maintaining unnecessarily redundant copies of large datasets.

Without an effective data management solution, the HPC effort will struggle with difficulties in staging data for analysis, recovering generated datasets, and inefficiencies created by insufficient physical storage and recomputing results that have once been completed. Strategically, the absence of an effective data management solution will present a barrier to supporting emerging efforts working to leverage the breadth of generated datasets for use in development of computationally and data intensive predictive models as well as efforts to utilize cloud resources for collaboration and analysis.

## HPC DME

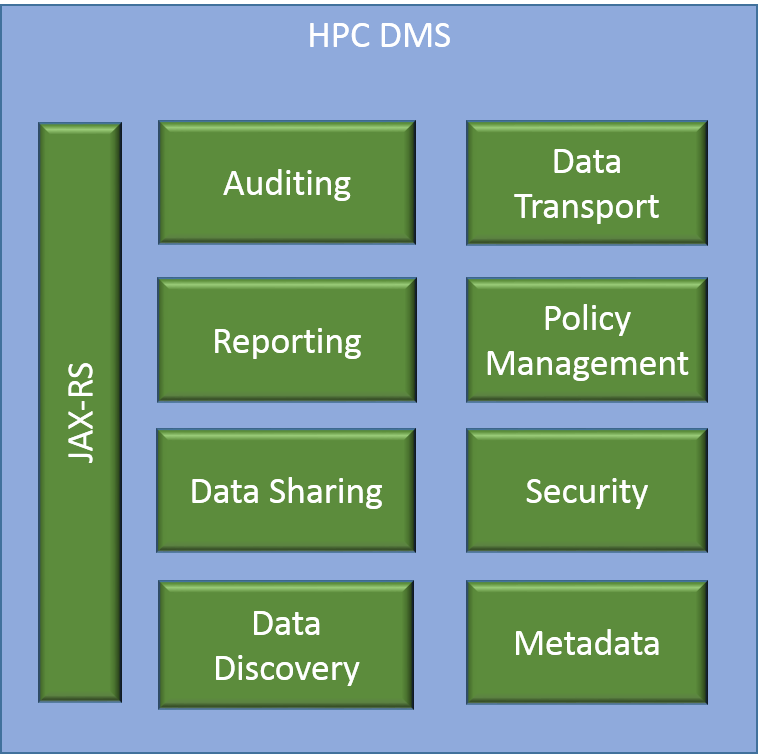
The HPC DME, High Performance Computing Data Management Environment, is a highly adaptable and an open ended data storage system supporting storage and management of huge amounts of data and its metadata. HPC DME provides capabilities for storing, managing, discovering, transferring and sharing huge amounts of data across collaborators securely and efficiently.

As shown in the diagram below, raw data generated from a sequencer, analyzed data generated by data analysis along with the metadata generated by LIMS or user defined metadata can securely and efficiently be stored in the HPC DM system. The metadata is stored in a reliable database with reference to the actual data in the archive system. This provides ability to decouple data and metadata so that the data archival storage and its underlying technology is transparent to the implementation.



Users can store data for a long term on HPC DME, share and transfer their data such that they do not have to redistribute or maintain copies of the data on other systems by eliminating the data integrity issues. HPC DME stores and associates user defined metadata to any registered data at different levels of data life cycle, enabling the system not only to help identify the data but also enhancing the search capabilities.

The HPC data management system provides a number of application programming interfaces (APIs) to operate and interact with it. These APIs acts as a facade for the underlying implementation to encapsulate and to provide consistent access to the users. The HPC data management system provides a high-reliability storage model for underlying datasets including a dataset registration system, and an API for transfer of large datasets with no-loss of data. By default, the HPC data management system integrates with Globus platform and uses it to perform data transfer tasks. The basic features of HPC data management system help users in registering and uploading their data to the HPC DM archive storage system and managing it. HPC DME archive storage can be a permanent storage for the users’ data and can be used as a platform to search, manage and transfer the data to other storage systems and also to share with other collaborators or users. Each data object is stored with its binary data along with its required and user defined metadata. The associated metadata can be used as search criteria to identify dataset(s).



Based on modular application development, HPC DM APIs are developed to provide flexible, pluggable and easily configurable application to support data management requirements. Each of the modules shown in the diagram can be extended or replaced easily without impacting the application. More details on these modules are discussed in the Software Architecture section (3.4) below. The pilot phase of this implementation is focused on integrating with iRODS for metadata management and with Globus for data transfer. With the modular architecture, any future integrations can be supported very easily.

### Metadata

Metadata is defined as the data about the data. It is the information which describes the actual data such as the date and origin of creation, its contents, its condition, processing it has gone through and associations to other objects etc. Metadata is employed to make data searches faster, more specific and also enable and promote data sharing among scientists.

HPC DME collects metadata for each data object/entity registered and stored in a database along with the associations. HPC DME collects two kinds of metadata related to a Collection or data object, namely, administrative and division/center specific. The administrative metadata is the required set of information which needs to be submitted at the time of registration with HPC DME. New metadata variables can be added to both administrative and division/center specific metadata sets after obtaining proper authorizations and permissions. The metadata can also be updated by authorized users. Apart from these two kinds of metadata, HPC DME also manages system generated metadata for each of the entities registered with the system.

One of the primary functions of HPC DME is to connect unstructured data with metadata. Metadata may be attached to files, folders and collections. HPC DME stores metadata in the form of “triples” to its relational database. The triples consist of an attribute field, a value field, and a unit field. The content of each of these fields can be independently defined and applied. This Metadata can be changed and updated through the life cycle of each data objects.

Adding metadata to objects or their collections can also be optional. A Division or center has flexibility to define their own metadata and its policies. One may choose not to define any metadata policies. Adding/updating policies at later time may require updates to existing metadata.

Following table shows default required Metadata for a collection in the HPC DME. This list can be modified by a system administrator to add, update or remove any of these attributes. Details on updating metadata attributes configuration is detailed in the section below.

**Metadata for Collection:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Metadata Variable** | **Definition** | **Default** |
| **Administrative Metadata** | | | |
|  | collection\_type | Collection type name (Default valid values are Project, Dataset, Folder). This is required attribute. |  |
|  | name | Name for the dataset of files as provided by the depositor |  |
|  | description | Description of dataset |  |
|  | source\_lab\_pi | PI of the lab of the depositor at the time of deposit |  |
|  | lab\_branch | Lab or Branch or Program the PI belongs to |  |
|  | pi\_doc | Division, Organization, Center the PI belongs to |  |
|  | original\_date\_created | Date the dataset was created originally | Date the dataset was deposited |
|  | data\_creator | Person or Organization lead who created the data | Not Specified |
|  | phi\_content | Presence of Protected Health Information in the datasets deposited via HPC DME.  Valid values are (PHI Present, PHI Not Present, Not Specified) | Not Specified |
|  | pii\_content | Presence of Personally Identifiable Information in the datasets deposited via HPC DME. Valid values are (PII Present, PII Not Present, Not Specified) | Not Specified |
|  | data\_encryption\_status | If the data is encrypted or not. Valid values are (Encrypted,  Not Encrypted, Not Specified) | Not Specified |
|  | data\_compression\_status | If the data is compressed or not | Not Specified |
|  | funding\_organization | Organization Funding the generation of Data | Not Specified |

Following table shows default required Metadata for a Project in the HPC DME.

**Metadata for Project:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Metadata Variable** | **Definition** | **Default** |
| **Administrative Metadata** | | | |
|  | collection\_type | Collection type name (Default valid values are Project, Dataset, Folder) |  |
|  | name | Name for the Project |  |
|  | description | Description of project |  |
|  | source\_lab\_pi | PI of the lab of the depositor at the time of deposit |  |
|  | lab\_branch | Lab or Branch or Program the PI belongs to |  |
|  | pi\_doc | Division, Organization, Center the PI belongs to |  |
|  | original\_date\_created | Date the dataset was created originally | Date the dataset was deposited |
|  | project\_type | Default valid values are "Unspecified", "Umbrella Project", "Sequencing", "Analysis" |  |
| **Division/Center specific metadata** | | | |
|  | Internal\_project\_id | Internal Project Id to track |  |
|  | comments | General text for internal use and reference |  |

**System Generated Metadata Variables for any Collection:**

|  |  |  |
| --- | --- | --- |
| Id | local identifier (serves as foreign key to connect to other metadata in data management system |  |
| uuid | universal dataset identifier (reserved for future use: default is ‘unspecified’) |  |
| create\_date | Date the project, dataset or file was registered with HPC DME. |  |
| update\_date | Date the project, dataset or file was updated with HPC DME. |  |
| registerd\_by | User depositing the dataset |  |
| registered\_by\_name | Name of the person registering the dataset |  |
| registered\_by\_doc | Division, Organization, Center of the user registering the dataset |  |

Following table shows default required Metadata for a data object in the HPC DME.

**Metadata for Data object/file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Metadata Variable** | **Definition** | **Default** |
| **Administrative metadata** | | | |
|  | name | Name for the file as provided by the depositor |  |
|  | description | Extensible description of File |  |
|  | source\_lab\_pi | PI of the lab of the depositor at the time of deposit | Auto Populated |
|  | lab\_branch | Lab or Branch or Program the PI belongs to |  |
|  | pi\_doc | Division, Organization, Center the PI belongs to |  |
|  | original\_date\_created | Date the File was created originally | Date the file was deposited |
|  | data\_creator | Person or Organization lead who created the data | Not Specified |
|  | phi\_content | Presence of Protected Health Information in the datasets deposited via HPC DME.  Valid values are (PHI Present, PHI Not Present, Not Specified) | Not Specified |
|  | pii\_content | Presence of Personally Identifiable Information in the datasets deposited via HPC DME. Valid values are (PII Present, PII Not Present, Not Specified) | Not Specified |
|  | data\_encryption\_status | If the data is encrypted or not. Valid values are (Encrypted,  Not Encrypted, Not Specified) | Not Specified |
|  | data\_compression\_status | If the file is compressed or not | Not Specified |
|  | funding\_organization | Organization Funding the generation of Data | Not Specified |
| **Division/Center specific metadata** | | | |
|  | comments | General text for internal use and reference |  |
| **System Generated Metadata Variables for any Data object / file (Automated)** | | | |
|  | id | local identifier (serves as foreign key to connect to other metadata in data management system |  |
|  | uuid | universal dataset identifier (reserved for future use: default is ‘unspecified’) |  |
|  | dataset\_signature\_md5 | MD5 checksum |  |
|  | Dataset\_signature\_sha1 | SHA1 checksum |  |
|  | Data\_size | Size of the dataset being deposited |  |
|  | create\_date | Date the project, dataset or file was registered with HPC DME. |  |
|  | update\_date | Date the project, dataset or file was updated with HPC DME. |  |
|  | registerd\_by | User depositing the dataset |  |
|  | registered\_by\_name | Name of the person registering the dataset |  |
|  | registered\_by\_doc | Division, Organization, Center of the user registering the dataset |  |
|  | source\_globus\_path | Data Source Globus Path |  |
|  | source\_globus\_endpoint | Data Source Globus Endpoint |  |
|  | Date\_globus\_endpoint | Data Location Globus Endpoint |  |
|  | data\_globus\_path | Data Location Globus Path |  |

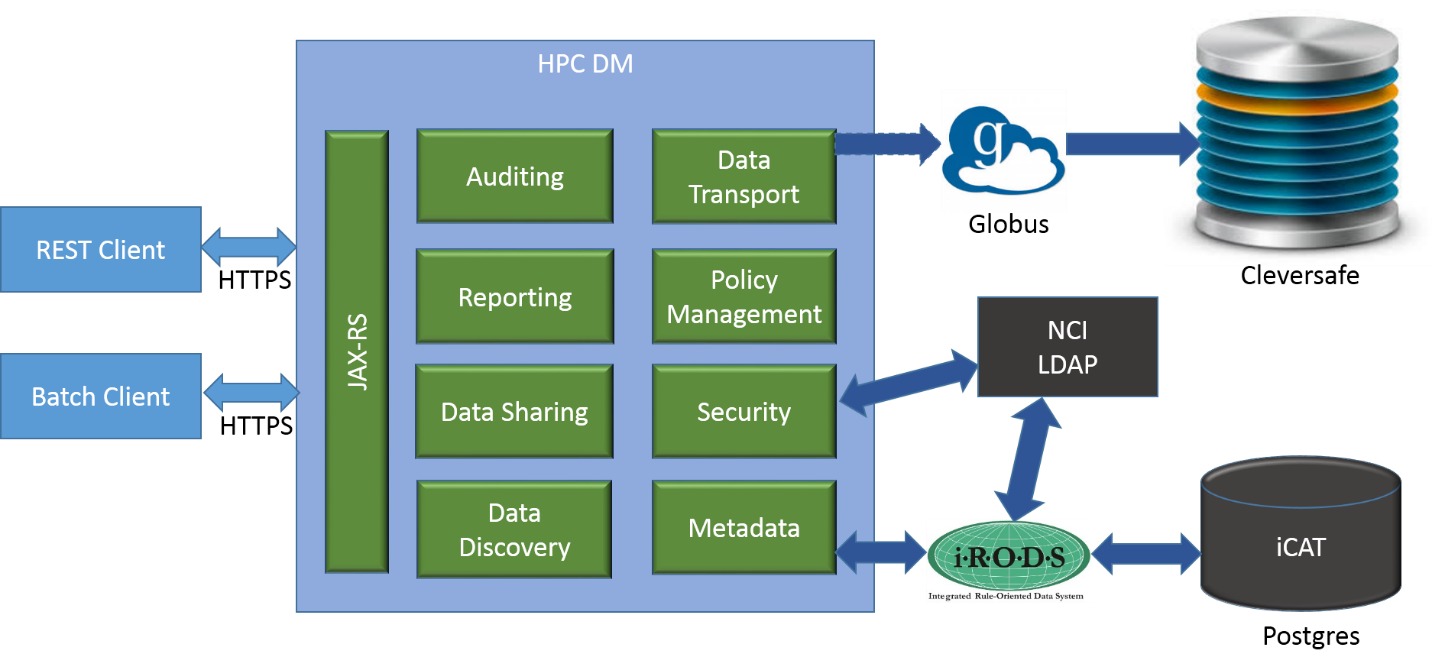
HPC DME provides flexible way to configure metadata attributes through a policy file. Following is an example of the policy file.

|  |
| --- |
| **Sample Policy File** |
| {"HpcMetadataValidationRules": {  "collectionMetadataValidationRules": [  {  "attribute": "Collection type",  "mandatory": true,  "validValues": [  "project",  "dataset",  "folder"  ],  "ruleEnabled": true,  "DOC": "DOC-NAME"  },  {  "attribute": "Project name",  "mandatory": true,  "collectionType": "project",  "ruleEnabled": true,  "DOC": "DOC-NAME"  },  {  "attribute": "Project type",  "mandatory": true,  "defaultValue": "Unspecified",  "collectionType": "project",  "validValues": [  "Umbrella Project",  "Sequencing",  "Analysis"  ],  "ruleEnabled": true,  "DOC": "DOC-NAME"  }, |

* attribute: Metadata attribute name
* mandatory: Flag to indicate if it is required or not. Valid values are “true”, “false”
* defaultValue: Default value of the attribute if no value is given
* collectionType: Collection type name applicable for this attribute rule. Only one value is allowed.
* validValues: List of valid values for this attribute
* ruleEnabled: Flag to indicate if this rule is enabled or not. Valid values are “true” or “false”. If value is set to “false”, this rule will not be evaluated during validation process.
* DOC: Division name applicable. This rule will only be active for the given DOC. This attribute is not active now.

### Data Archival

The center piece of HPC DME is to reliably store large data and be able to retrieve it when needed. HPC Data transfer API abstracts the underlying implementation to the end user so that the transfer and Storage technologies can be extended or replaced as needed. By default, HPC DME uses Globus API to perform data transfer between Globus endpoint asynchronously. Globus API manages data movement tasks to a hosted service that manages the entire operation, monitoring performance and errors, retrying failed transfers, correcting problems automatically whenever possible, and reporting status to keep users informed.



When a data object registration request is submitted to HPC DME through its interface, after successful validation, data object metadata is registered with metadata repository and a data transfer request is initiated with Globus asynchronously. As part of the registration request, caller would have to provide valid source endpoint and location of the file.

"source": {

"endpoint": "nihnci#NIH-NCI-TRANSFER1",

"path": "/GridFTP/GridFTP\_t3/FNLCR\_LAB/ProjectX/DatasetY/62-acd0-089ea9370450/test5.fastq"

}

"destination": {

"endpoint": "nihfnlcr#gridftp1",

"path": "/mnt/gridftp"

}

Destination endpoint information is optional. If destination information is not given, data object is written into HPC DME default home location along with the given path. The default archive destination endpoint and the path are both configurable through system configuration.

Once the data transfer request is submitted to Globus, its transfer status can be one the following states:

|  |  |
| --- | --- |
| INITIATED | The transfer of a dataset is initiated. |
| IN\_PROGRESS | The transfer of a dataset is in progress. |
| SUCCEEDED | The transfer of a dataset has been successful and is complete. |
| FAILED | The transfer of a dataset is not successful when globus fails to transfer files due to errors like file not found etc., after repeated attempts. |

When a data request is submitted to Globus, HPC DME API collects initial response of the request and store it in HPC DME database. This response information is updated in following ways to present up to date information to the users when querying.

1. Cron job running at scheduled intervals query for all requests NOT in SUCCEEDED or FAILED state with Globus to get latest status and update them in HPC DME. If any data objects are in FAILED state, its registration will be rolled back.
2. When querying for data object, if it’s state is not SUCCEEDED, HPC DME would query Globus to get latest status and update it. If the data object is in FAILED state, its registration will be rolled back.

Following are the details of data transfer response captured in HPC DME:

Task ID                : 7ea479c0-ba3d-11e5-9a07-22000b96db58  
Task Type            : TRANSFER  
Status                    : SUCCEEDED  
Is Paused              : No  
Request Time         : 2016-01-13 21:34:59Z  
Deadline                 : 2016-01-14 21:34:59Z  
Completion Time    : 2016-01-13 21:35:02Z  
Total Tasks             : 3  
Tasks Successful    : 3  
Tasks Expired          : 0  
Tasks Canceled       : 0  
Tasks Failed             : 0  
Tasks Pending          : 0  
Tasks Retrying         : 0  
Command                : API 0.10  
Label                    : n/a  
Source Endpoint Name     : nihnci#NIH-NCI-TRANSFER1  
Destination Endpoint Name: nihfnlcr#gridftp1  
Source Endpoint          : e1c6b3bd-6d04-11e5-ba46-22000b92c6ec  
Destination Endpoint     : dabdccc3-6d04-11e5-ba46-22000b92c6ec  
Sync Level               : n/a  
Data Encryption          : No  
Checksum Verification    : No  
Delete                   : No  
Files                    : 1  
Files Transferred        : 1  
Files Skipped            : 0  
Directories              : 1  
Expansions               : 1  
Bytes Transferred        : 30629739  
Bytes Checksummed (Sync) : 0  
Effective MBits/sec      : 81.679  
Faults                   : 0

### Data Discovery

Data discovery in HPC DME enables users to locate the data in the archive system either by metadata or its logical path.

**Discovery by metadata:**

The data discovery functionality enables users to search for collections or files they want to download, transfer or share with other researchers and collaborators, utilizing the associated metadata as search terms. This kind of search based on metadata, improves the search functionality by tremendously expanding the search criteria according to the user specifications and not just the technical aspects of the dataset.

Metadata can be queried by its name, value and a comparator. Following operators are supported to query by attribute value:

* EQUAL
* NOT\_EQUAL
* LESS\_THAN
* GREATER\_THAN
* LESS\_OR\_EQUAL
* GREATER\_OR\_EQUAL
* LIKE

**Discovery by path:**

HPC DME supports data virtualization by decoupling its logical path with its physical storage page. When a data object or collection is registered with HPC DME, it is registered with a logical path. For example: /FNLCR\_LAB/ProjectX/DatasetY. This provides flexibility for users to organize their data in a structured format irrespective of the physical storage location and its path. HPC DME supports querying these data objects or collections by their logical path.

## Data Sharing

Data sharing is critical part of data collaboration. Through HPC DME, users can share a collection or data file with another user. When a user registers a collection or data object, the user is by default becomes OWNER of that collection or data object. User may assign OWN, READ, WRITE permissions to another user. Assigning NONE permission will take away any existing permissions. These permissions are by default cascades down with in the data hierarchy. If you assign a permission on a collection, that permission will be recursively inherits to all its child collections and objects. You may override this behavior by manually changing the permission on child collection or objects. Any permission updates done at higher level will always overwrite any manual permissions at lower level.

## Audit

TBF

## policy management

TBF

## reports

HPC DME API supports generating following reports. These reports are generated by directly querying underlying metadata repository.

* Summary report: Summarized report on entire archive storage.
* Summary report by data range: Summarized report on entire archive storage based on date range.
* DOC Report: Summarized report for a selected DOC
* DOC Report by date range: Summarized report of selected DOC for a selected date range
* User Report: Summarized report for a selected User
* User Report by date range: Summarized report of selected User for a selected date range

## Hardware Architecture

[Insert any related hardware architecture documents or provide a reference to where they are stored.]

## Software Architecture

HPC DME system architecture includes following 4 tiers:

1. **API Server:** A server that will host a collection of components and APIs for the various services provided by the system, and provide workflow capabilities and integration with other sub-systems.
2. **Data Management:** An integrated data management system – iRODS.
3. **Data Transfer:** A data transfer service - Globus
4. **Authentication:** Authentication service via NCI LDAP server.

The proposed architecture, defined by the above components, is designed to address the following key project objectives:

* Effective data management while supporting a variety of dataset formats and managing metadata, annotations, cataloging.
* The flexibility to allow for the evolution of associated dataset formats over time.
* High-performing dataset discovery.
* Reliable, scalable, and high performing data storage and data transfer.
* The application of security best practices for both authentication and authorization by integrating with NCI services.

### API Server

The HPC-DM API server architecture is a multilayered architecture to support clear separation of concerns and boundaries among the various components of the server. The architecture includes following 6 horizontal layers.

1. REST Web service layer
2. Batch layer
3. Business service layer
4. Application service layer
5. Integration layer
6. DAO layer

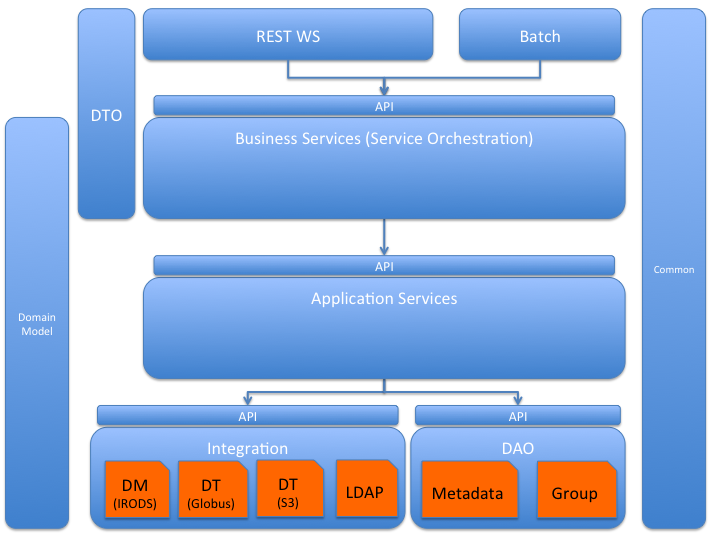
Each horizontal layer is comprised of 2 OSGi bundles (dynamically loadable JAR), one contains the layer’s API and the other contains the API’s implementation. A layer’s API bundle is configured (via a manifest file) to export (or make available) the API interface packages. A layer’s implementation bundle is configured to hide all of its code. This approach has the following advantages:

1. It ensures layers are communicating with each other via APIs since the implementation is physically not visible outside the bundle
2. It ensures avoiding circular dependencies among the bundles and thus promotes the architecture reusability.
3. It enables a hot deployment of implementation code, i.e. swapping a layer’s implementation bundle in runtime without stopping the server.

The architecture also includes following 3 vertical layers, which include POJO code that is used by multiple horizontal layers to communicate.

1. Domain layer
2. DTO Layer
3. Common layer

The following diagram depict the API server architecture:



#### REST Web Service Layer:

This layer is responsible to define and implement the RESTful Web Services API. It is responsible for:

* Implementing each REST API by invoking the appropriate Business Service.
* Deploying secured (SSL) and non-secured endpoints.
* User (Service Invoker) authentication and authorization
* Mapping Service Input from JSON/XML to DTO POJO
* Mapping Service Output to JSON/XML and HTTP return code.
* Attach API version to all services response.

#### Server Batch

This layer is responsible for implementing task scheduler, invoking various server’s batch processes. Each scheduled task is implemented by invoking business services.

#### Business Services

This layer is responsible for defining and implementing coarse-grained business services which are the system’s API defined in pure Java. This layer is responsible for:

* Map DTO (The POJO data types used to define the business services API and are mapped to JSON/XML) to Domain Model (The internal POJO used by the system and are not exposed to the services invoker)
* Implementing each coarse-grained business service by orchestrating a series of application services.

#### Application Services

This layer is responsible for defining and implementing fine-grained application services that are used as building blocks to deliver business services. This layer is responsible for:

* Domain Model and Data validation rules
* Implementing application services by applying business logic and invoking API offered by the Integration and DAO layers..

#### Integration Layer

This layer is responsible for defining and implementing an API to the subsystems the server is integrated with. The API is defined generically and completely hides any integrated system details, so the underlying system can be easily replaced if needed. This layer is responsible for:

* ‘Data Management’ API implemented via Jargon API to integrate with iRODS
* ‘Data Transfer’ API implemented by REST web services to integrate with Globus
* ‘Authentication’ API implemented with Spring LDAP to integrate with NCI LDAP server.

#### DAO Layer

This layer is responsible for persisting data to the database. The current implementation persists data to PostgreSQL DB.

#### Data Transfer Object (DTO) & Domain Model

These vertical layers define POJO data objects that are used in communications between the layers. The DTO are Java representation of data objects used in external API. Domain model are data objects used to implement services and business logic.

#### Common

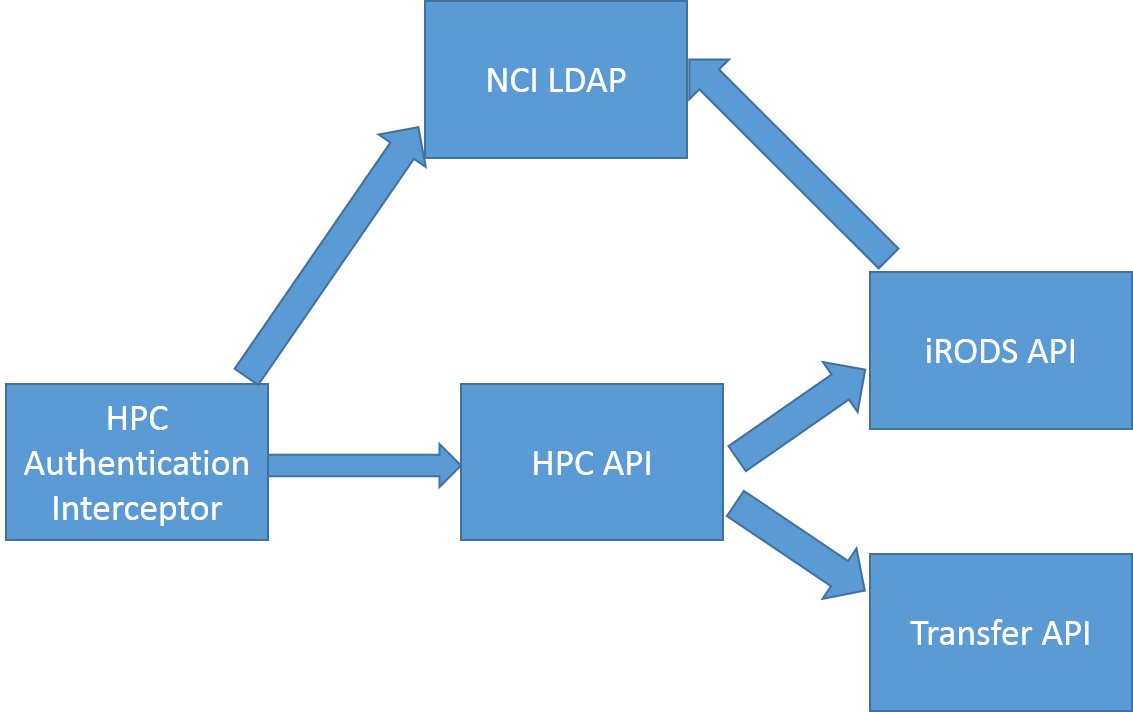
Common includes utility Java code that is used in all horizontal layers, and also include the system exception classes.

## Security Architecture

HPC DME operations are secured by authentication and authorization.

**Authentication:**

HPC DME authenticates with NCI Active Directory to authenticate a user. HPC DME security integrates with Globus and iRODS to securely interacts with their respective APIs. Every call to HPC API is intercepted by HPC Authentication Interceptor to validate user credentials with NCI LDAP. Once the user is authenticated with HPC API, any subsequent call to iRODS API or Data transfer API would need to be authenticated as well. With the current state of integration with iRODS, a user would need to be authenticated by iRODS system for secure access.

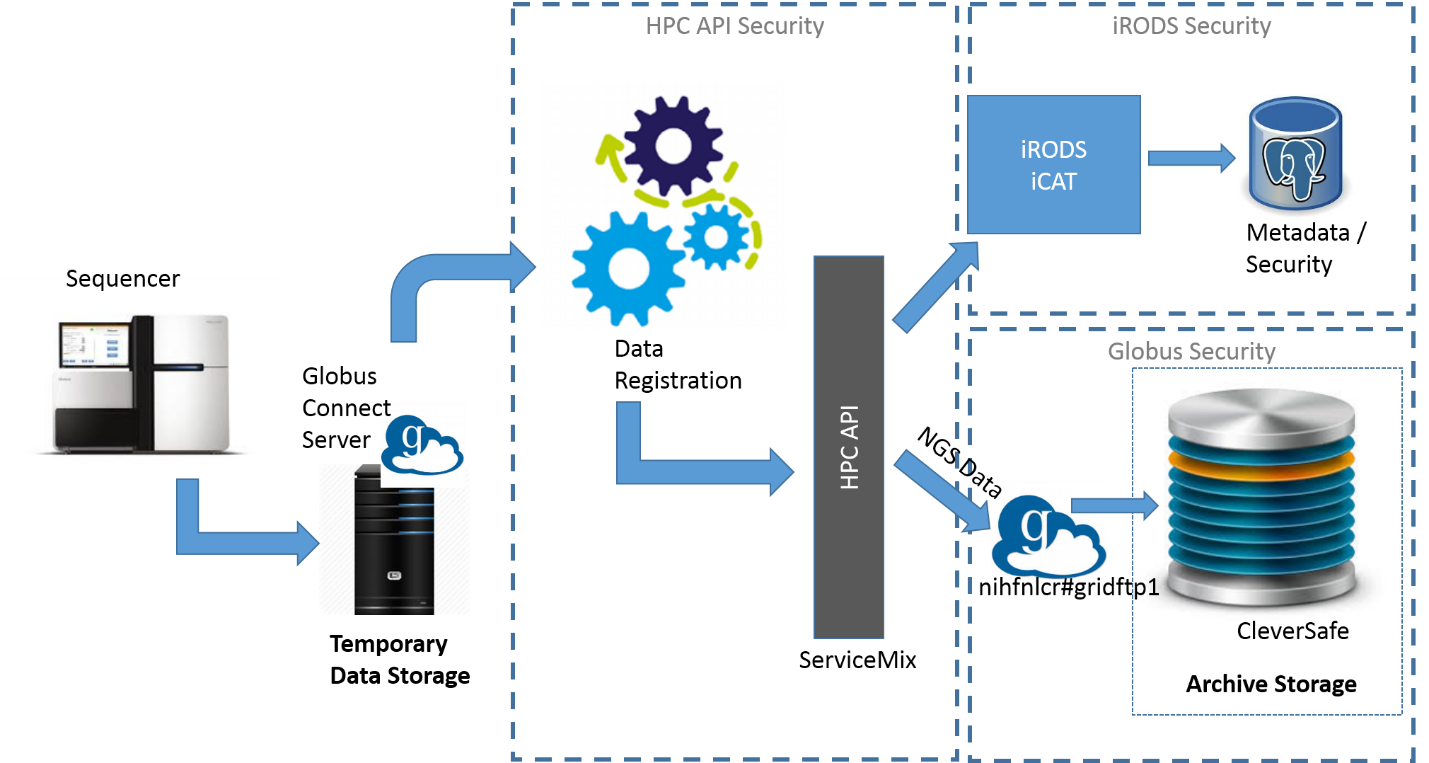


By default, iRODS uses a secure password system for user authentication. The user passwords are scrambled and stored in the iCAT database. Additionally, iRODS supports user authentication via PAM (Pluggable Authentication Modules), which can be configured to support many things, including the LDAP authentication system. PAM can be configured to support various authentication systems; however, the iRODS administrator still needs to add the users to the iRODS database. For this pilot implementation, iRODS is configured to use PAM authentication with NCI LDAP. A pluggable authentication module (PAM) is a mechanism to integrate multiple low-level authentication schemes into a high-level application programming interface (API). It allows programs that rely on authentication to be written independently of the underlying authentication scheme.

Globus is the default data transfer mechanism with the HPC DME pilot implementation. Current release of Globus API requires Globus account to be created to start using its data transfer feature. During user registration with HPC DME, user would need to provide Globus credentials and these credentials are stored in HPC DME database in an encrypted manner. These stored credentials are used to authenticate with Globus to initiate any data transfer. Future release of Globus is promised to remove the dependency of having Globus account to initiate data transfer. That would enable single-sign-on approach with HPC DME and Globus integration.

Users of HPC API interface are required to enroll into HPC system to start using HPC system. A System Administrator would enroll a user with HPC DME with NCI userId. HPC DME would use same userId to create an account with iRODS when registering with HPC DME. When a user account is created in iRODS, iRODS will create logical name with the userId. By default, the user owns this logical space on iRODS.

HPC Server API is configured to support BASIC authentication credentials over HTTPS. Given credentials are verified against NCI LDAP for authentication. Authenticated User is verified with HPC security system for authorization.



**Authorization:**

HPC DME authorization covers authorizing users with HPC API and iRODS features only. HPC DME authorization does not cover or extend Globus authorization policies at this time.

HPC Server APIs are protected by user roles extended from iRODS. Through Spring configuration these authorization settings on APIs can be updated easily. HPC Server API internally interacts with iRODS Jargon API to work with iRODS securely. HPC DME encapsulates iRODS authorization policies.

iRODS primarily provides three primary roles,

* + Admin - can manage user accounts and access permissions on different objects and collections.
  + User - owner of collections or objects can assign different access types to different users on the owned collections or objects.
  + Groupadmin - can create users, groups and add or delete users from groups

iRODS provides the following access types:

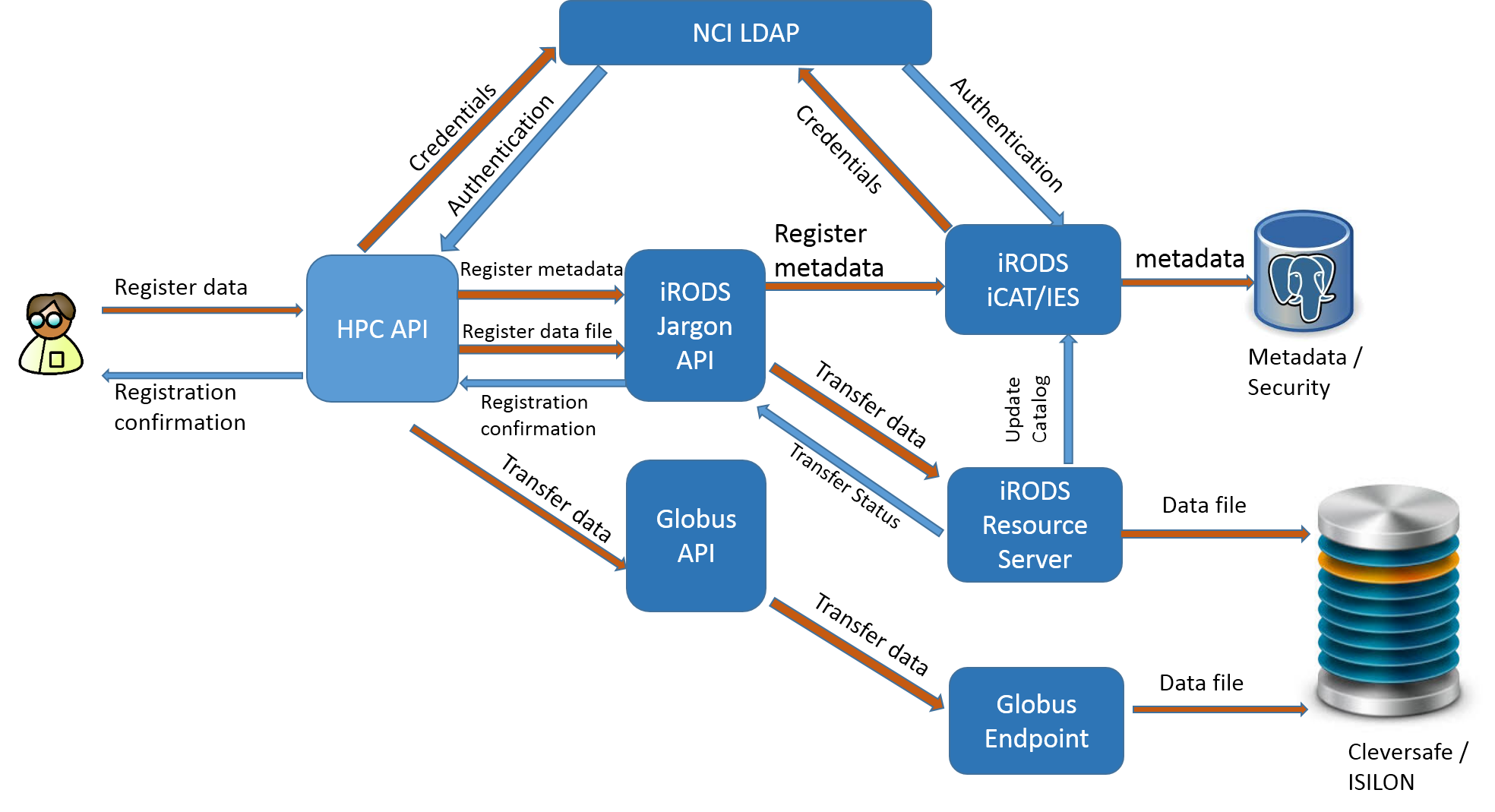
* + Own
  + Read
  + Write
  + None

Subsequent phases of HPC DME implementation would extend iRODS authorization model to provide more granular roles and access types. It would also provide single-sign-on capability where authenticating with NCI would seamlessly get access to any integrated module like iRODS, Globus, Cleversafe etc.

## Communication Architecture

Following diagrams depicts communication workflow of different systems involved with HPC DME. A sequencer or data creator would keep its data in one of its Globus endpoints. A user registered with HPC DME system would invoke HPC API with credentials to register the data with the archive system. User credentials are authenticated with NCI LDAP to store given metadata into HPC DME database. HPC API would also initiate data transfer through Globus API asynchronously.

Users of HPC DME interacts with HPC REST API over HTTPS protocol. It is expected to have HPC API running on ServiceMix and iRODS iCAT instance hosted on same machine to eliminate extra security configuration and setup needs for iRODS.



## Performance

[Insert any performance documents or provide a reference to where they are stored.]

# System Design

## Database Design

HPC DME adopts iRODS data model to manage metadata and security around that. Each iRODS deployment—or Zone—is composed of an iRODS Metadata Catalog (iCAT), an iCAT-Enabled Server (IES), and optional Resource Servers. The iCAT is a relational database that holds all the information about data, users, and zone that the iRODS servers—IES and any resource servers—need to facilitate the management and sharing of any data registered with HPC DME. The iCAT contains the information about

• the zone for the purposes of sharing across zones,

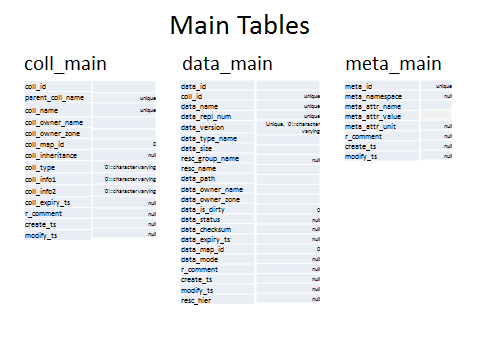
• data and their metadata,

• the virtual file system,

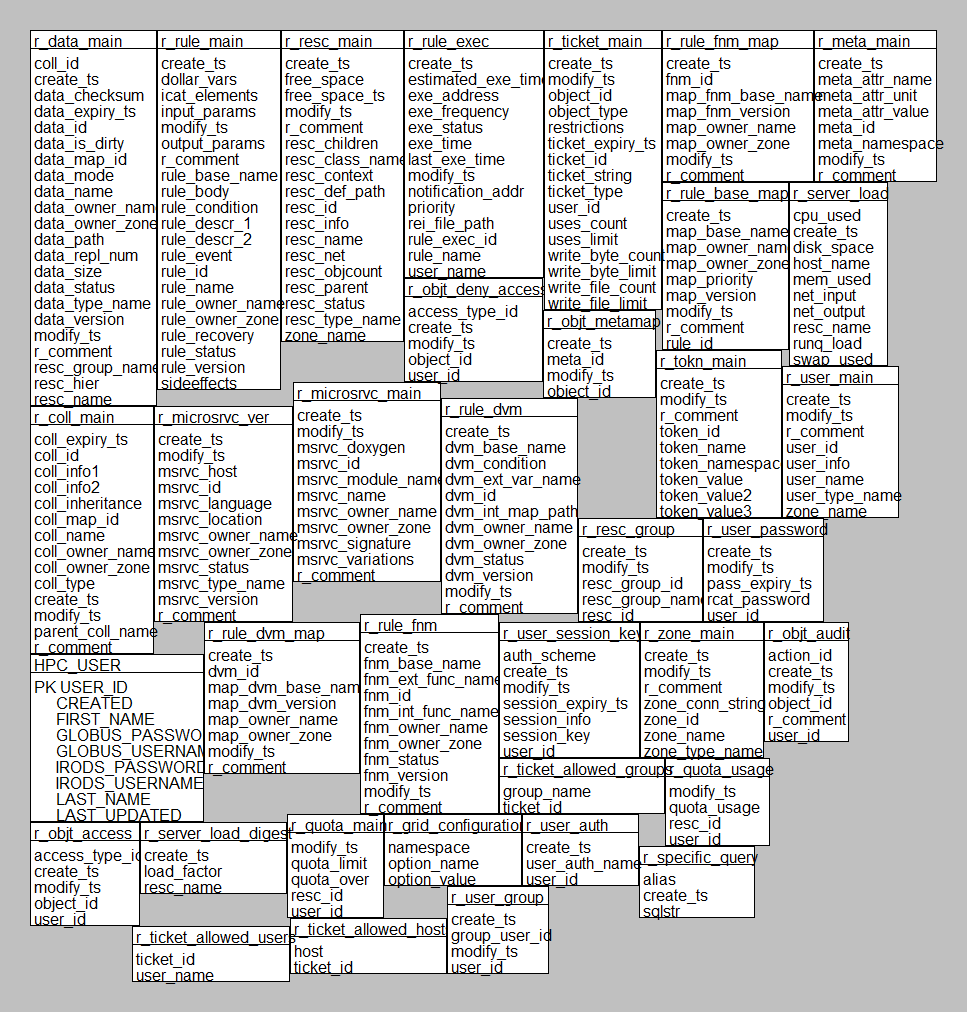
• resource configuration, and

• user information.

The iRODS data elements are grouped into several tables. The most important tables which map to HPC requirements are the tables representing Collections, Data Objects and the Metadata. These tables are the coll\_main, data\_main and meta\_main. These tables are shown in the image below. The image also shows the fields which can have a ‘Null’ value and which ones can’t. The collections and the data tables are connected to the metadata table by the map\_id column. Metadata may be attached to files, users, groups, collections (iRODS equivalent of sub-directories), and resources.

****

Following diagram shows HPC DME schema.



## Application Program Interfaces

Please refer to API specification document at <https://ncisvn.nci.nih.gov/svn/HPC_Data_Management/branches/hpc-prototype-dev/doc/guides/HPC_Server_API.docx>

## System Integration

The HPC DM API server provides a set of data management REST services to deposit, annotate, search and retrieve large data objects. The API is implemented by combining capabilities of several external systems. This document provides the technical details on the how the external systems integration was implemented and lists the resulted dependencies.

### Integrated Systems

The HPC DM API Server is integrated with the following external Systems:

**iRODS**   
An open source data management solution. iRODS provides data registration with metadata catalog. It provides data search, security, audit, rule-engine and data transfer capabilities. (<https://irods.org>).

The integration with iRODS was developed using the Jargon API (<https://github.com/DICE-UNC/jargon/wiki/Jargon-overview>). There are several requirements that could not be satisfied with Jargon, and the server is querying the iRODS DB directly.

**Globus**   
An open source data transfer and sharing platform. Globus provides ability to transfer large files asynchronously, securely and reliably. (<https://www.globus.org/>)

The integration with Globus was developed using the Globus transfer API using Globus nexus API to authenticate w/ Globus. Both are Java jars from Globus that are packaged with the API server. The transfer API provides convenient Java API to communicate with the Globus REST services (https://docs.globus.org/api/transfer).

**Cleversafe**

IBM’s object storage solution, which is deployed on-premises but can be deployed to public or dedicated cloud. Note that Cleversafe supports the Amazon S3 data transfer API. (<https://www.ibm.com/cloud-computing/products/storage/object-storage/>).

The integration with Cleversafe was developed using the Amazon AWS SDK for Java (<https://aws.amazon.com/sdk-for-java>).

**Active Directory**

Microsoft’s directory and authentication system. The integration was done using the Java standard naming API, and is trivial so not covered in this document.

### Code Organization

The integration code can be found in the following maven projects and is organized in the following java packages:

* **hpc-integration-api**
  + *gov.nih.nci.hpc.integration* – package contains all integration interfaces (This is the pure Java API the ‘application services’ are using to invoke the integration services)
* **hpc-integration-impl**
  + *gov.nih.nci.hpc.integration.globus.impl* – Globus implementation of the Data Transfer Integration API
  + *gov.nih.nci.hpc.integration.s3.impl* – S3 implementation of the Data Transfer Integration API .
  + *gov.nih.nci.hpc.integration.irods.impl* – Implementation of the Data Transfer Management Integration API
  + *gov.nih.nci.hpc.integration.ldap.impl* – LDAP implementation of the Security Authentication Integration API .

The iRODS integration includes direct querying of iRODS DB. The relevant source code can be found in the following maven project

* **hpc-dao-impl**
  + *hpc\_hierarchical\_metadata.sql* – This is a script to create a set of materialized views based on iRODS table. HPC DM provides a capability to search data object by metadata defined anywhere in the object ‘hierarchy’ (i.e. metadata that are associated with the containing collection hierarchy tree). The purpose of the materialized view is to generate that ‘hierarchical metadata’ table, so search queries can be easily implemented.
  + *HpcMetadataDAOImpl.java* – Implements the data search by querying the hierarchical metadata materialized view.
  + *HpcGroupDAOImpl* – HPC DM provides a capability to search for user groups using case insensitive matching. This capability is not available via the Jargon API, so the implementation queries the iRODS table directly.

### IRODS Integration

#### Jargon API Integration Points

Jargon is a Java API provides a convenient way to interact with iRODS. The current Jargon version used is 4.1.10.0. The following table lists the integration points that are implemented with Jargon and the specific API that was used:

| **Integration Point** | **Jargon API** |
| --- | --- |
| authenticate | IRODSAccessObjectFactory.*authenticateIRODSAccount*()  AuthResponse.*getAuthenticatedIRODSAccount*() |
| disconnect | IRODSAccessObjectFactory.*closeSessionAndEatExceptions*() |
| createCollectionDirectory | IRODSFileFactory.*instanceIRODSFile*()  IRODSFile.*mkdirs*() |
| createDataObjectFile | IRODSFileFactory.*instanceIRODSFile*()  IRODSFile.*createNewFile*() |
| delete | IRODSFileFactory.*instanceIRODSFile*()  IRODSFile.deleteWithForceOption() |
| addMetadataToCollection | CollectionAO.*addBulkAVUMetadataToCollection()* |
| updatCollectionMetadata | CollectionAO.modifyAvuValueBasedOnGivenAttributeAndUnit*()* |
| addMetadataToDataObject | DataObjectAO.*addBulkAVUMetadataToDataObject()* |
| updatDataObjectMetadata | DataObjectAO.modifyAvuValueBasedOnGivenAttributeAndUnit*()* |
| getPathAttributes | IRODSFileFactory.*instanceIRODSFile*() |
| getCollection | CollectionAO.*findByAbsolutePath()*  CollectionAndDataObjectListAndSearchAO.*list DataObjectsAndCollectionsUnderPath()* |
| getCollectionMetadata | CollectionAO.findMetadataValuesForCollection*()* |
| getDataObject | DataObjectAO.*findByAbsolutePath()* |
| getDataObjects | DataObjectAO.*findDomainByMetadataQuery()* |
| getCollectionMetadata | DataObjectAO.findMetadataValuesForDataObject*()* |
| addUser | UserAO.*addUser()* |
| deleteUser | UserAO.*deleteUser()* |
| updateUser | UserAO.*updateUser()* |
| getUserRole  userExists | UserAO.*findByName()* |
| getCollectionPermissions | CollectionAO.listPermissionsForCollection*()* |
| getCollectionPermissionForUser | CollectionAO.getPermissionForUserName*()* |
| setCollectionPermission | CollectionAO.setAccessPermission*()* |
| getDataObjectPermissions | DataObjectAO.listPermissionsForDataObject*()* |
| getDataObjectPermissionForUser | DataObjectAO.getPermissionForDataObjectForUserName*()* |
| setDataObjectPermission | DataObjectAO.setAccessPermission*()* |
| addGroup | UserGroupAO.*addUserGroup()* |
| deleteGroup | UserGroupAO.*removeUserGroup()* |
| groupExists | UserGroupAO.*findWhere()* |
| addGroupMember | UserGroupAO.*addUserToGroup()* |
| deleteGroupMember | UserGroupAO.*removeUserFromGroup()* |
| getGroupMembers | UserGroupAO.*listUserGroupMembers()* |

## 

#### Materialized Views Integration Points

To support hierarchical metadata for collections and data objects, 6 materialized views were created. The following tables depict the dependencies of the materialized views on iRODS DB tables:

|  |  |
| --- | --- |
| **HPC DM Materialized View** | **Dependent iRODS tables** |
| r\_coll\_hierarchy\_matamap | r\_coll\_main  r\_objt\_metamap |
| r\_coll\_hierarchy\_metamain | r\_coll\_hierarchy\_metamap  r\_meta\_main  r\_objt\_metamap |
| r\_coll\_hierarchy\_meta\_attr\_name | r\_coll\_hierarchy\_metamap |
| r\_data\_hierarchy\_matamap | r\_data\_main  r\_objt\_metamap |
| r\_data\_hierarchy\_metamain | r\_data\_hierarchy\_metamap  r\_meta\_main  r\_objt\_metamap |
| r\_data\_hierarchy\_meta\_attr\_name | r\_data\_hierarchy\_metamap |

The following table lists the integration points that are implemented with the materialized views / iRODS DB

|  |  |
| --- | --- |
| **Integration Point** | **Materialized View / iRODS DB table** |
| getCollectionPaths  getCollectionCount | r\_coll\_hierarchy\_meta\_main  r\_objt\_access  r\_user\_main  r\_user\_group |
| getDataObjectPaths  getDataObjectCount | r\_data\_hierarchy\_meta\_main  r\_objt\_access  r\_user\_main  r\_user\_group |
| getCollectionMetadata | r\_coll\_hierarchy\_meta\_main |
| getDataObjectMetadata | r\_data\_hierarchy\_meta\_main |
| getCollectionMetadataAttributes | r\_coll\_hierarchy\_meta\_attr\_name  r\_objt\_access  r\_user\_main  r\_user\_group |
| getDataObjectMetadataAttributes | r\_data\_hierarchy\_meta\_attr\_name  r\_objt\_access  r\_user\_main  r\_user\_group |
| getGroups | r\_user\_main |

### Globus Integration

### Cleversafe Integration

## User Interface Design

[Insert any user interface design documents or provide a reference to where they are stored.]

## System Performance

[Insert any system performance documents or provide a reference to where they are stored.]

Appendix A: Design Specification Approval

The undersigned acknowledge that they have reviewed the **HPC DME Design Specification** and agree with the information presented within this document. Changes to this **Design Specification** will be coordinated with, and approved by, the undersigned, or their designated representatives.

[List the individuals whose signatures are desired. Examples of such individuals are Business Owner, Project Manager (if identified), and any appropriate stakeholders. Add additional lines for signature as necessary.]

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: | Zhengwu Lu |  |  |
| Title: | Technical Project Manager |  |  |
| Role: |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: |  |  |  |
| Title: |  |  |  |
| Role: |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: |  |  |  |
| Title: |  |  |  |
| Role: |  |  |  |

APPENDIX B: REFERENCES

[Insert the name, version number, description, and physical location of any documents referenced in this document. Add rows to the table as necessary.]

The following table summarizes the documents referenced in this document.

|  |  |  |
| --- | --- | --- |
| **Document Name** | **Description** | **Location** |
| *HPC API Specification* |  | <https://ncisvn.nci.nih.gov/svn/HPC_Data_Management/branches/hpc-prototype-dev/doc/guides/HPC_Server_API.docx> |
| *HPC DME Requirements* |  | <https://ncisvn.nci.nih.gov/svn/HPC_Data_Management/branches/hpc-prototype-dev/doc/requirements/HPC_requirements_definition.docx> |
|  |  |  |

APPENDIX C: KEY TERMS

The following table provides definitions and explanations for terms and acronyms relevant to the content presented within this document.

|  |  |
| --- | --- |
| **Term** | **Definition** |
| DTO | Data Transfer Object |
| DAO | Data Access Object |
| HPC DME | High Performance Computing Data Management System |
| API | Application Program Interface |