Science & Technology Facilities Council

Data Download (and Upload)

NOBUGS 2012 ICAT Workshop

27th September 2012

Kevin Phipps

Scientific Computing Department, RAL

Overview

- Handling files in ICAT
- Why a data server is needed
- Proposal for a Data Server Interface Specification

Where are the data files?

- The files themselves are NOT stored in ICAT
- In ICAT a Datafile ‘location’ field is typically relative path to the actual file on disk eg.
  `Investigation123/dataset456/datafileName.ext`
- The problem:

  - Dangerous to open up an area of the file system to all users of ICAT

  - Not practical to have each ICAT user set up as a user on the OS and control file permissions that way
- A ‘Data Server’ component is needed

Data Server

- Currently a custom “Data Server” needs to be written to store, retrieve and delete the files and to communicate with ICAT for authorisation purposes
  - requests sent directly to data server
  - authorisation checked with ICAT
  - file stored, removed or returned
  - ICAT Datafile entry added or removed

Existing Data Servers

- 4 facilities at RAL all have their own implementations
- TopCat needs a ‘download manager’ written to interface with each of these data servers
- A common specification is highly desirable
- A fairly generic ICAT Data Server (IDS2) is in use on one of our projects. We intend to modify it to meet the specification and make it available as a reference implementation.

Data Server Interface Specification

- Includes calls to:
  - store individual data files
  - retrieve, query status of, delete groups of data files
- Does not define how groups of files will be returned (facilities may have specific requirements on this)
- Recommendation for zip file format which will be included in reference implementation

ICAT Coupling

- ICAT session ID passed as argument to most calls
- For data retrieval requests the data server must check for **R**ead permissions in ICAT
- For put and delete requests it must check for **C**reate and **D**elete permissions and make corresponding changes in ICAT
- Consistency with ICAT must be maintained – orphan file preferred to an ICAT entry with no corresponding file

Web Service Style

- Web service specification follows “REST” guidelines
- HTTP methods PUT, DELETE, POST and GET used
- POST supported in addition to GET for requests where URLs would become too long
- Calls taking parameters investigationIds, datasetIds and datafileIds use a comma separated list

Dual Storage Model

- Based on assumption that there is a (fast) local cache of recently used data and a (slower) archive system
- Implementation needs to manage the cache of local files
- If one storage device is fast enough and large enough to hold all the data then this complexity is not required (but still follow the model)

1) Upload a file to data server

- **put**

- http method: PUT
- request header fields: sessionId, name, location, description, fileSize, doi, checksum, datafileCreateTime, datafileModTime, datafileFormatId, datasetId
- return: string representation of the id of the created datafile

- The body of the servlet request is the contents of the file to be stored
- Implementation also registers the file in ICAT
- datafileModTime and datafileCreateTime must be in the format YYYY-MM-DD hh:mm:ss
- **C**reate permission is required

Science & Technology Facilities Council

2) Delete file(s) from data server

- **delete**

- http method: DELETE
- parameters: sessionId, investigationIds, datasetIds, datafileIds

  - Deletion of investigations, datasets and datafiles causes the contained components to be deleted both from the dataserver and from ICAT

  - Delete permission is required

3) Get / download file(s)

- **getData**

- http method: GET
- parameters: sessionId, investigationIds, datasetIds, datafileIds, compress
- return: the requested datafile or datafiles.

  - If some or all of the data are not on fast storage the implementation should commence retrieval of the data and attach an exit code to the response.

  - compress may have the value 1 to indicate a high degree of compression, 0 to indicate none and if omitted the level of compression is implementation defined.

4) Remove file from local storage

- **archive**

- http method: POST
- parameters: sessionId, investigationIds, datasetIds, datafileIds

  - Archiving of investigations, datasets and datafiles is a hint that the datafiles may be moved to storage where access may be slower

  - Read permission is required. This is because there is no risk to the data, an archiving request can at most delay access.

  - For a single fast storage setup this method needs no implementation

5) Make files available on local storage

- **restore**

- http method: POST
  parameters: sessionId, investigationIds, datasetIds, datafileIds

  - Restoration of investigations, datasets and datafiles is a hint that the datafiles be moved to storage where access is faster
  - **R**ead permission is required.
  - For a single fast storage setup this method needs no implementation

6) Get status of data

- **getStatus**

- http method: GET
- parameters: investigationIds, datasetIds, datafileIds
- return: a string with:

  - ONLINE if all requested items of data are available on fast storage
  - RESTORING if data has been requested but is not available yet
  - ARCHIVED if data is not on the fast storage and has not been requested

  - This does not require a sessionId so no permissions are required.
  - For a single fast storage setup ONLINE can always be returned

Summary

- ICAT does not store the actual files
- Some kind of data server needs to be implemented
- Needs to respect ICAT permissions
- A reference implementation will be available soon
- If all implementations follow the common specification then tools using multiple ICATs will benefit
- Only 6 methods to implement (3 if using one single storage solution)