

# Unlucky for Some

## The Thirteen Core Use Cases of HEP Metadata

Version 1.0: December 2004

Tim Barrass

Peter Kunszt, Gavin McCance, Krzysztof Nienartowicz, Ricardo Rocha

Randolph Herber, Adam Lyon, Wyatt Merrit

Mòrag Burgon-Lyon, Tony Doyle, **Steven Hanlon\***, Paul Millar, Richard St. Denis

Solveig Albrand, Jerome Fulachier

James Werner

Carmine Cioffi, Stefan Stonjek

Stephen Burke, Owen Synge

*Bristol*

*CERN EGEE project*

*Fermilab*

*Glasgow*

*LPSC, Grenoble*

*Manchester*

*Oxford*

*RAL*

## Objective

Metadata is commonly defined as 'data about data'. More specifically, it is ancillary information about stored data, which aids a user in handling it, describing it and understanding what it contains.

The objective of this document is to survey all available HEP use case documents, select from them the use cases which are common to Grid-like computing systems and identify the implications on the metadata catalogue which is to be used.

In this document, only physicist-level use cases are considered; that is, only the requirements of the end user. Furthermore, the term 'metadata' is taken to refer high-level information about jobs, physics events and collections of data (or 'datasets'). In particular, it does not include file and replica information. It is assumed that some lower-level catalogue will exist to perform this function, as laid out in the ARDA document [1].

## Introduction

This document identifies use cases which can be considered fundamental to a Grid-like computing system for a HEP experiment and which imply requirements on the metadata catalogues used. High-energy physics use case and requirements documents have been surveyed and core use cases harvested.

Information from several experiments has been used to compile this document:

- from the LHC experiments, the HEPCAL I [2] and II [3] documents;
- the CDF use case document, CDF5858 [4];
- "Catalog Services for ATLAS" from the ADA group in ATLAS [5]; and
- "BaBar Analysis Grid Application – Use Case and Requirements Document" [6].

Summaries of these documents have been included here as appendices. In addition, a glossary of terms is included for reference.

---

\* Primary contact person for this document.

## Datasets

When dealing with a computer file system, it is traditional to handle data in units of 'files'. However, in a Grid-like system, the concept of a file, a block of data treated as a unit by file systems, need not concern a user. Instead, a user can consider collections of data which are conceptually linked and unencumbered by restrictions from lower-level systems (on size, for example), while transparent middleware handles the files themselves. These collections are called 'datasets'.

This is a very general description of a dataset. There are many, very specific (and often mutually exclusive) definitions which pin down the properties of these datasets. In this document, however, only two general classes of definition will be considered:

- **logical dataset** – a collection containing a particular group of logical files. This kind of dataset is defined by the particular data it contains. Datasets are defined in this way in HEPCAL. Metadata can then be added to describe the contents of the dataset. (In the Fermilab SAM system [7], this kind of dataset is called a 'snapshot' because it is the result of resolving the contents of a virtual dataset at a particular point in time.)
- **virtual dataset** – a recipe for creating a logical dataset. The definition specifies the properties of the data that it contains, rather than referring to particular logical files. For example, a virtual dataset could be defined as all the output from a program when run over an input dataset, or all data satisfying a search query. Datasets are defined this way in the SAM system.

In the use cases below, datasets are referred to frequently. Where it is important whether a dataset is logical or virtual, this distinction has been made. Note that use cases which demand a logical dataset as an input could also take a virtual dataset, as long as a logical copy of it is produced first.

## Use Cases

The identified common use cases are described below. This list is not intended to be complete but to provide an overlapping set of use cases. Much of the richness in detail presented by the contributing documents is absent. It is instead a proposed common ground for all experiments to meet on and use as a basis for collaboration.

The use cases have been divided into three categories:

- Dataset Handling – use cases related to the creation and access of data and metadata.
- Analysis – use cases dealing specifically with the needs of physics analysis.
- Job Handling – use cases for submitting, querying and controlling jobs submitted to a computing resource.

The use cases are described from the user's standpoint and in terms of metadata implications. Each is related to the appropriate use cases in the source documents (see appendices). Table 1 summarises these relationships.

### 1. Dataset Handling

#### 1.1 Specify a new dataset.

The user registers the results of a process as a logical dataset. Alternatively, the user might specify a virtual dataset by describing a process to produce the new dataset. Often the process will simply be a search in the metadata catalogue which will select data to be included in the

dataset, but it could be more complicated (e.g. an analysis executable to run over the data, or a Monte Carlo program to produce new simulated data). The user also specifies the metadata to be associated with the dataset.

### *Metadata Implications*

It must be possible to specify the metadata in bulk at the creation time of a dataset. If the new dataset is derived from a previous one, the relationship should be recorded.

### *Source Use Cases*

HEPCAL: dsreg

CDF: 4.2

ADA: UC1.1, UC2.1, UC2.4, UC3.1, UC4.1, UC4.3, UC4.4

## 1.2 Read metadata for datasets.

The user searches through datasets, identifying all that satisfy a given metadata query, or he browses all the information available on a given dataset.

### *Metadata Implications*

Must be able to query metadata quickly for searches. User must be able to uniquely identify a dataset. To aid searching, both string- and numerical-type metadata should be allowed. A date type (with a defined time-stamp standard) would also be helpful.

### *Source Use Cases*

HEPCAL: dsmdacc

CDF: 4.1

ADA: UC1.5, UC5.3

BABAR: U1.5

## 1.3 Update metadata for a dataset\*.

A user or administrator identifies a dataset and either alters metadata currently associated with it or adds further information. This use case also encompasses the use of custom user metadata, perhaps in the form of data collections that do not refer to files directly, instead containing references to datasets or their contents (a user catalogue).

### *Metadata Implications*

User must be able to uniquely identify a dataset. It must be possible to evolve metadata over time. To implement user catalogues inside the metadata system, it should be possible to create a data collections that does not relate to files, but to datasets, and possibly smaller objects, in catalogues.

### *Source Use Cases*

HEPCAL: dsmdupd

---

\* This use case is discussed in more detail later, under *User Models and Altering Metadata*.

ADA: UC5.3 (also supports attaching the results of data quality checks, as per UC1.4 and UC1.5)

CDF: Dataset metadata cannot be changed, but data can be added to a dataset.

## 1.4 Resolve physical data.

The user has selected the catalogue entry for a dataset of interest and wishes to locate the data described. If the dataset is virtual, processing must be done to determine which files should be contained in the dataset, thus creating a logical version of the dataset.

### *Metadata Implications*

The book-keeping system must hold a reference which is meaningful to the system which deals with data storage and which will allow physical files associated with the dataset to be located. It would also be helpful if the metadata allowed datasets with no physical data files (e.g. virtual datasets, datasets which refer only to other datasets) to be easily identifiable.

### *Source Use Cases*

HEPCAL: dsaccess, remdsacc

CDF: 4.6, 4.7, 4.8

ADA: UC5.1 (also data quality checks: UC1.4 and UC1.5.)

BABAR: U1.6

## 1.5 Access data in a dataset.

The user requests and obtains some sort of resource locator which can be used to access data associated with a logical dataset. This could be, for example, a path and physical file name or an FTP URL. This may be used for downloading data, or for accessing it from a job on the user's local machine.

### *Metadata Implications*

The metadata should include some indication of the size of the logical dataset referred to. It would be helpful if it were possible to access metadata for downloaded datasets. For fully offline use of the dataset, it must be possible to download the metadata and query it locally.

### *Source Use Cases*

HEPCAL: dsdownload

CDF: 4.5, 4.9

## 2. Analysis

### 2.1 Run a physics simulation program.

The user specifies a Monte Carlo program executable and a set of parameters with which it is to run. The program then runs, placing its output in a new logical dataset if desired.

### *Metadata Implications*

The user must be able to specify the parameters with which to run the Monte Carlo. These

parameters, as well as a link to the executable used, must be stored with the metadata for the output dataset so that it can be identified later.

#### *Source Use Cases*

HEPCAL: simjob

CDF: 4.4, 4.10, 4.11

BABAR: Noted in Further Use Cases

## 2.2 Select a subset of a dataset.

The user constructs a metadata query and it is applied to the input logical datasets. A new logical dataset is produced consisting of all data satisfying the query. The query may act on the input dataset's constituent datasets, files, events or some mixture. The copy may be shallow (i.e. output references to the contents of other datasets' files) or deep (i.e. duplicate the data in new files) , for more efficient access.

#### *Metadata Implications*

It must be possible to apply a query at dataset, file and event levels. It must be possible to create a logical dataset that holds references to datasets, files or events. It must be possible to record the relationship between the input and output datasets.

#### *Source Use Cases*

HEPCAL: analysis1

CDF: 4.7

ADA: UC4.2, UC4.4

BABAR: U1.5

## 2.3 Run an algorithm over an input dataset.

The user defines an input logical dataset, an executable, a configuration and, possibly, an output dataset. The executable realises an algorithm which is applied to all data in the input dataset. The output from the algorithm may then be used to produce a new dataset, perhaps a sub-set of the input, e.g. selection of data using variables not available in the metadata catalogues, or otherwise a super-set, i.e. output consists of an existing dataset plus extra information.

#### *Metadata Implications*

The metadata catalogue must be able to store the executable and configuration, and how they relate the input and output datasets. Requirements are as for case 2.1, above.

#### *Source Use Cases*

HEPCAL: dstrans, analysis1

CDF: 4.6, 4.9

ADA: UC1.4, UC1.5, UC2.2, UC2.3, UC2.4, UC5.1

BABAR: Use Case 1

### 3. Job Handling

#### 3.1 Submit a job to a Grid.

A user describes a job in terms of input logical datasets, local input files, program to run, output files, environment required and optional additional attributes and submits it to the Grid. The information is stored and the program is run to completion. This could also apply to some non-Grid computing resource.

##### *Metadata Implications*

There must exist a catalogue which stores all the appropriate information about a given job, including the executable, all inputs and outputs and configuration information and any restrictions on where the job may run.

##### *Source Use Cases*

HEPCAL: jobsubmit

CDF: 4.3

BABAR: U1.9

#### 3.2 Retrieve/Access the output of a job.

The user identifies a previously submitted job and examines the output. The output might be a Grid-registered logical dataset which can be queried as above (“Read metadata for datasets”) or it may be an ordinary file, which the user may download to his local machine.

##### *Metadata Implications*

Jobs must be uniquely identifiable so that they can be located at a later date. The job catalogue must retain information about output Grid-registered logical datasets so that they can be identified. A job may need to be re-run if a logical dataset needs to be reproduced (if a job fails or for confirmation purposes) so it is necessary for all the metadata that was required for the job to run (see use case 3.1) be retained after the job terminates. Job information should be retained some time longer than output datasets exist, and possibly forever\*.

##### *Source Use Cases*

HEPCAL: joboutaccess

CDF: 4.9

BABAR: U1.14

#### 3.3 Estimate the system resource cost of running a job.

The user describes a job, as above, and asks the system to estimate the resource cost of running it. This estimate will include the cost of accessing all necessary datasets (and creating logical copies of virtual datasets), as well as the cost in processor time of running the executable.

---

\* The exact length of time that job information should be retained depends on the timescale over which the implementing experiment might require a job to be repeated. It should, however, probably be of the months to years scale, rather than hours or days.

### *Metadata Implications*

The job catalogue must contain sufficient information to make the running cost estimate. Therefore, it must be possible to create a job without immediately submitting it to any computing resource.

### *Source Use Cases*

HEPCAL: jobresest

CDF: Noted in 'Middleware Requirements', MW6

## 3.4 Monitor the progress of a job.

The user identifies a previously submitted job which is not yet complete and obtains information on the running of the job, including the location it is being run at, the time that the job has been running already and any estimate of the remaining running time that is available. The user is then able to issue commands, such as holding, resuming or cancelling the job. The system should also alert the user to any processing failures that have taken place.

### *Metadata Implications*

As noted in use case 3.2, a job must be uniquely identifiable so that it can be located later. It must be possible to connect an entry in the job catalogue to the job itself on the computing resource and use that connection to interrogate the computing resource about the progress of the job.

### *Source Use Cases*

HEPCAL: jobcont, jobmon

CDF: Noted in 'Middleware Requirements', MW9

BABAR: Use Case 2

## 3.5 Repeat a previous job.

The user identifies a previously submitted job which he wishes to run once again. This may be because the job failed in running or because the user is checking previous analysis work. He may also wish to change the job parameters before re-running because he has found a bug in his executable or configuration.

### *Metadata Implications*

Again, a job must be uniquely identifiable. All the proper configuration parameters must be stored. It must be possible to create a new job using a previous job from the catalogue as a starting point.

### *Source Use Cases*

HEPCAL: See HEPCAL II on 'Provenance and Job Traceability'

CDF: 4.6

BABAR: Use Case 4 and Use Case 5.

Table 1: The Thirteen Core Use Cases for HEP Metadata.

		Core Use Case	Source Use Cases			
			HEPCAL	CDF	ATLAS	BaBar
Dataset Handling	1.1	Specify a new Dataset	dsreg	4.2	UC1.1, UC2.1, UC2.4, UC3.1, UC4.1, UC4.3, UC4.4	
	1.2	Read Metadata for Datasets	dsmdacc	4.1	UC1.5, UC5.3	U1.5
	1.3	Update Metadata for a Dataset	dsmdupd	Not Allowed	UC5.3	
	1.4	Resolve Physical Data	dsaccess, remdsacc	4.6, 4.7, 4.8	UC5.3	U1.6
	1.5	Download a Dataset to Local Disk	dsdownload	4.5, 4.9		
Analysis	2.1	Run a Physics Simulation Program	simjob	4.4, 4.10, 4.11		
	2.2	Select a Subset of a Dataset	analysis1	4.7	UC4.2, UC4.4	U1.5
	2.3	Run an Algorithm over an Input Dataset	dstrans, analysis1	4.6, 4.9	UC1.4, UC1.5, UC2.2, UC2.3, UC2.4, UC5.1	U1
Job Handling	3.1	Submit a Job to the Grid	jobsub	4.3		U1.9
	3.2	Retrieve/Access Output of a Job	joboutaccess	4.9		U1.14
	3.3	Estimate the Resource cost of Running a Job	jobreest	MW6		
	3.4	Monitor the Progress of a Job	jobcont, jobmon	MW9		U2
	3.5	Repeat a Previous Job	See HEPCAL II	4.6		U4, U5

## User Models and Altering Metadata

In addition to defining use cases as above, it is also necessary to specify the extent to which the system will offer different privileges to different users.

The simplest scheme simply defines a dataset, and its associated metadata, as a write-once read-many object and allows all registered users to read all datasets and write new datasets, but prohibits any changes to the metadata of established datasets. The CDF and D0 collaborations have implemented this approach in their SAM data access application and have shown that, despite its simplicity, this is a practical system for use in a running experiment. However, it is not without problems. If a dataset is created with the wrong metadata, it is not possible to change it; a new dataset must be created with a new, distinct name and the correct metadata. This is a problem particularly when the dataset is being created to be used by many end users and has been named well. The name is forever reserved for the flawed dataset.

There have been few detailed proposals on how a more complex system would work. HEPCAL suggests that users would gain privileges through 'roles' distributed by their Virtual Organisation (VO). In particular, it specifies a hierarchy of analysis: End User, Group Level and Production. How this would relate to metadata privileges is not clear.

However, at first glance, the requirements of the metadata system are not over-complex. A simple scheme can be proposed:

- An Administrator has full read and write access to the metadata.
- A Project Administrator has full read access and limited write access to a defined part of the database.
- A User has full read access and can write new datasets, but cannot alter existing ones.

It should be noted that, in order to be able to reproduce the state of a dataset at any time, it must be possible to retrieve the previous value of an item of metadata after it has been changed. A good mutable metadata system would record all changes, who made them, when and why and would allow the history to be inspected and used.

## Conclusions

All available HEP use case documents have been surveyed and the most fundamental use cases have been identified and described. A general set of HEP metadata operations has been defined and these form a common ground for developers to work from.

This document is not intended to be exhaustive; areas in which further work would be useful are easy to identify. The three most important are:

- Input from further HEP experiments;
- Consideration of lower-level use cases; and
- More detailed examination of user roles and the alteration of metadata.

It is proposed that metadata developers, whether designing a metadata solution for a hypothetical new experiment or adding functionality to an existing experiment, should examine these use cases and use them as a basis for developing more detailed requirements.

## References

- 1: P. Buncic et al., [Architectural Roadmap Towards Distributed Analysis](#), 2003
- 2: F. Carminati et al, [Common Use Cases for a HEP Common Application Layer - HEPCAL](#), 2002
- 3: D. Barberis et al, [Common Use Cases for a HEP Common Application Layer for Analysis - HEPCAL II](#) , 2003
- 4: B. Todd Huffman et al, [The CDF/D0 UK GridPP Project](#), 2002
- 5: David Adams, [Catalog Services for ATLAS](#), 2004
- 6: [BaBar Analysis Grid application - Use Case and Requirements Document](#), 2001
- 7: [The SAMGrid Project](#), <http://projects.fnal.gov/samgrid/>
- 8: David Adams, [AJDL: Analysis Job Description Language](#), 2004

## Glossary

Book-keeping – Storage of definitions of datasets, and information associated with them.

Catalogue – The combination of some sort of data store and a well-defined interface which allows a client to access it.

Dataset – A set of data grouped together for some purpose.

Job – A user-defined process run on some computing resource.

Logical Dataset – A Dataset defined by the data objects it contains, be they datasets, files, events or other objects. Contrast to Virtual Dataset.

Logical File – On a Grid-like system, there may be several identical copies of a file in various

physical locations. These are all said to be physical replicas of the same logical file.

Metadata – Ancillary information about stored data. For the purposes of this document, the term is reserved for information regarding jobs, datasets and events.

User Catalogue – A data collection which does not refer to files directly, instead containing references to Datasets or their contents. This could simply be a flat file holding references to datasets, events or files.

Virtual Dataset – A book-keeping entry which holds a recipe for producing a Logical Dataset. The recipe could be a search query or a more complex specification of an algorithm.

## Appendices

The following pages present summaries of the five contributing use case documents. The *Source Use Cases* listed above can be found here. However, these are provided for reference only; the reader is referred to the original documents for greater detail.

## HEPCAL-I

### Introduction

HEPCAL-I seeks to identify implementation-independent use cases and requirements for computing Grids for the LHC experiments. It distinguishes between two classes of job: organised and chaotic. Organised jobs have well-defined input which is accessed sequentially and output which is typically some transform of the input. In general the focus for these jobs is on total throughput, rather than response time. Production and Reconstruction are examples of this class of job. In contrast, Analysis tends to be more chaotic. HEPCAL-I focuses on organised jobs, leaving the problem of analysis to HEPCAL-II.

### Common Ground - Definitions and Requirements

The concept of a dataset is introduced. A dataset is any collection of information, such as raw or processed events or sets of histograms. A dataset may be composed of other datasets. A software package can also be considered a dataset, if it is managed by the Grid. Datasets are registered to the Grid with a unique logical dataset name (LDN). Once registered, a dataset is fixed.

Assumed common characteristics are described. These are:

- *Data Structures* - raw data, Monte Carlo data, event summary data (reconstructed information + enough to redo reconstruction), analysis object data (physics objects), event tags (brief information for first pass selection).
- *Event Identifiers* - there exists a method of unambiguously identifying an event.
- *Mapping events to datasets* - there will be a method of getting from an event identifier to the datasets containing information on the event.
- *Identifying Datasets* - datasets have a unique, user-chosen LDN. There will be a catalogue of metadata on each dataset, in the form of keyword-value pairs.
- *Event Metadata* - considered application-dependent, dealt with by the experiments.
- *Conditions Data* - there will be a database of running conditions and calibration data.
- *Event Independence* - events can be processed in any order.

- *Access Permissions* - there will be a method for restricting rights to datasets.
- *Job Information* - basic job identifiers, composite job identifiers (e.g. process chains, split jobs), production job identifiers. Should be a job catalogue, containing keyword-value pairs.

## Use Cases

A set of actors which will take part in computing activity are defined. For each of these, a set of high-level activities are defined, forming an outline of the use cases. Examples of these actors and their activities are:

- *User* - run algorithms on data, build private collections of data, fill histograms, make selections, run simulations.
- *Production Manager* - process all events of a certain type to a new data sample, data quality checks, submit and monitor job progress.
- *Database Manager* - publish new detector/beam conditions data, check conditions data, publish Event Summary Data and Analysis Object Data.
- *Experiment Resource Manager* - allocate resources (disk, CPU) to groups/individuals.
- *Manager* - Physics Coordinator approving production and monitoring production progress; Computing Coordinator supervising and altering resource allocations,
- *Software Developer* - test software on Grid.
- *Software Distributor* - release and register new package version.

Use cases are presented, organised by purpose. Not all the HEPCAL use cases are reproduced here, just the main ones.

## General

- Typically low level e.g. Grid log on.
- **Browse Grid Resources** (gridbrowse)

## Data Management

- Deal with user access to datasets. Users access datasets by requesting an LDN and receiving a physical file identifier. Data management should track access patterns in order to replicate intelligently.
- **Dataset registration to the Grid** (dsreg)
- **Dataset access** (dsaccess/remdsacc)
- **Dataset metadata update/access** (dsmdupd/dsmdacc)
- **Dataset upload** (dsupload)
- **Dataset download** (dsdownload)
- **Dataset delete** (dsdelete)
- **Dataset browsing** (dsbrowse)
- **Browse conditions database** (cdbbrowse)
- **User-defined catalogue create/delete** (catcrea/catdelete)
- **Dataset access cost evaluation** (dsaccesscost)
- Additional use cases are lower level e.g. dataset replication, verification, registration to the Grid.

## Job Management

- **Job catalogue update** (jobcatupd)

- **Job catalogue query** (jobcatquery)
- **Job submission** (jobsubmit)
- **Job output access/retrieval** (joboutaccess)
- **Error recovery** (jobrecov)
- **Job Control** - suspend, cancel, resume etc. (jobcont)
- **Job Monitoring** (jobmon)
- **Job resource estimation** (jobresest)
- **Dataset transformation** (dstrans)
- **Analysis** (analysis1)
- **Simulation Job** (simjob)
- Additional lower level use cases e.g. job splitting, environment changes, resource estimation.

## Virtual Organisation Management

- Sketchier use cases are provided for this area. Deals with the configuration of groups e.g. privileges, quotas, job catalogue options, adding/removing users, software publishing.

## Future Work

Many issues are not addressed. The most important areas for future work are:

- Catalogue requirements, which are not complete;
- Interactive Grid work;
- Job splitting.

## HEPCAL-II

### Introduction

HEPCAL-I focused on the problems of well-organised activities such as reconstruction and Monte Carlo production. HEPCAL-II supplements this with an examination of the problems and requirements of analysis activity.

### Analysis Characteristics

Analysis consists of iterative loops over some subset of the following steps:

1. Perform queries on dataset metadata to **identify datasets of interest**;
2. Query datasets with event-level metadata to **identify events of interest**;
3. **Save** results of step 2 for future use;
4. Perform **analysis activity** on selected events. Such activity could be:
  - additional filtering;
  - reprocessing;
  - addition of new information on events;
 or other actions;
5. **Save** results of step 4 and **publish** them somehow. Output can be:
  - shallow copy e.g. tag dataset
  - deep copy for more efficient access

- new dataset; necessary for adding new information to a dataset.

It is important that a user should be able to see some estimate of the resource cost of a procedure before committing to it.

The important characteristics of analysis activity, from the computing point of view, are identified:

- non-standard software - user code;
- input not known a priori;
- sparse data access pattern;
- large amount of concurrent, uncoordinated submission;
- interactive jobs;
- requirements on response time (as opposed to total throughput);
- detailed provenance information required;
- resource estimator required; and
- resources must be limited and accounted for.

## Notes on Metadata

- HEPICAL-II distinguishes between event-level metadata and dataset-level metadata.
- One can also have metadata for components of an event e.g. sub-detector or physics channel the component is relevant to.
- Metadata is to be used for provenance and book-keeping.
- Queries can be split into two sub-queries. One is on the dataset-level catalogue, the other is at event level, and is carried out in experiment-dependent software.

## Workload Management

Four workload management concepts are discussed:

- **No workload management** - user program sends the job to a computing element, which runs the query, accesses the data and runs analysis code.
- **Single job at best computing element** - a query on the dataset catalogue is made to determine the best computing element selection. That element then executes the event-level query on a supplied list of input datasets and executes analysis code.
- **Multiple jobs with merged output** - a query is made on the dataset catalogue. The job is split between multiple computing elements, and the outputs of these sub-jobs are merged at the end of execution.
- **Multiple queries with merged input** - a query is made on the dataset catalogue. The event-level query is then distributed to multiple computing elements which select events of interest. Selected events are then merged into an input dataset for the analysis code, running on a single computing element.

It is also important to note that analysis puts demands on **response time**. The iterative nature of the activity makes it important to be able to get fast responses.

## Requirements

HEPCAL-II describes in detail four requirements specific to analysis work: availability of the provenance of datasets, the logging of analysis activities, persistent interactive sessions and deployment of user analysis code.

## Provenance

It should be possible to relate a dataset to entries in the job metadata catalogue regarding the jobs used to produce it. One should be able to retrieve information on the executable name, environment settings and input files. This should allow the output to be reproduced. In this way, it should be possible to ban particular datasets, implicitly banning derived datasets.

One problem with this is determining the minimal amount of data which must be stored to allow the above. It is suggested that it may be best done by pooling metadata from various sources e.g. job catalogue, dataset catalogue, computing element configuration log.

## Logging

The system should log the results of any task not immediately discarded as useless in enough detail to make them repeatable. If the provenance information requirements above are fulfilled, this should just consist of compiling and annotating that information. The system should also allow querying of the status of current tasks. These logs should be usable both by groups working together and by individuals.

## Persistent Interactive Sessions

It should be possible to save an interactive session to the Grid and restore it later from elsewhere, maintaining all open datasets, loaded libraries and environment settings, query results, histograms etc.

## Analysis Software Deployment

User code may rely on particular libraries, compilers, linkers, environment variables, shells or other factors. There needs to be some mechanism to make sure that the computing element that will run the job has the proper set up. A possibility is to consider dependencies as input datasets so that the workload management system can take them into account when allocating resources to the job.

## Use Cases

Use cases in HEPCAL-II are presented differently to those in HEPCAL-I. Whereas, in HEPCAL-I, the tendency was toward detail, HEPCAL-II presents only three use cases:

- Production Analysis;
- Group-level Analysis; and
- End user Analysis.

These differ in terms of permissions and in what is done with the output i.e. end user analysis output is downloaded to a user's local store, whereas production and group-level analyses' results in new, metadata described, datasets for input into further analyses.

## Future Work

Two areas identified for future work are:

- requirements on private metadata; and
- resource allocation to individuals and groups.

## CDF Use Cases

### Introduction

CDF5858 describes CDF's requirements of a Grid-like system for data access and resource sharing. Use cases are presented.

### CDF Data Model

CDF data are divided into datasets, organised by trigger type. The analysis chain starts with **Raw Data**, which are processed to produce **Secondary Data Sets**, consisting of higher-level objects, such as tracks and jets, in addition to the raw data. These are then filtered to produce **Tertiary Data Sets**. These are a subset of the secondary data sets, and may contain less information per event. Finally, **Ntuples**, which can be processed on local machines, are produced for individual analyses.

### Requirements

Requirements on the middleware and data file catalogue are outlined. A basic assumption is that the standard CDF software will be available at all sites, though the possibility of packaging dependencies with the binary for distribution is noted.

#### Middleware Requirements

- A web-based tool for generating reports on datasets. (MW1)
- The ability to add a new dataset to the catalogue. (MW2)
- That data may be copied to/from a particular site, and the appropriate metadata be updated. (MW3)
- That the database may be used to track where data is available. (MW4)
- That sites publish available system resources. (MW5)
- A facility which determines the cost of an operation to the end user. (MW6)
- Determination of priority by matching resources needed on a given computing node to resources available. (MW8)
- Job status monitoring. (MW9)
- A method of determining the most cost effective manner to store data across multiple sites or at one site. (MW11)
- That storage resource permissions may be set up. (MW12)
- A database of simulation parameters. (MW14)
- Middleware should conform to Fermilab security policy. (MW16)

#### Data File Catalogue Requirements

The data file catalogue must contain:

- data location information (DFC1);
- persistence state e.g. disk or tape (DFC2);
- reprocessing information e.g. time and type of processing, software version (DFC3);
- provenance (DFC4); and

- all generator parameters for Monte Carlo files (DFC5).

## Use Cases

High-level use cases are presented based on the actions that an end user would wish to initiate. These are:

- 4.1 **Identify a dataset**
- 4.2 **Specify a new dataset**
- 4.3 **Submit a job**
- 4.4 **Specify simulation parameters**
- 4.5 **Populate local disks with data**
- 4.6 **Reprocess data**
- 4.7 **Skim to create a compressed sample**
- 4.8 **Create an ntuple**
- 4.9 **Use an ntuple locally**
- 4.10 **Run simulation**
- 4.11 **Run fast simulation** - only very simple detector simulation.

## The CDF L3 Trigger

It is noted that the CDF L3 Trigger contains functionality useful to a Grid-based system. It includes a database relating software versions to executables. Libraries and dependencies can be packaged for distribution. Finally, database constants can be exported to a flat-file format for local use.

## CDF Goals

The document describes three stages of functionality. At stage 1, job submission and data handling are largely manual. At stage 2, data handling is more automated, and there exist tools for determining CPU availability and job monitoring. At stage 3, the system is Grid-like, with jobs being automatically submitted to computing nodes depending on resource conditions.

Plans for the use of the SAM system in CDF, and for Grid enhancements to SAM, are then described in more detail than is appropriate here.

## Catalog Services for ATLAS

### Introduction

The document 'Catalog services for ATLAS' describes use cases which define requirements on data catalogues, in the context of the Analysis Job Description Language (AJDL) model [8].

## Catalogue Services

Data is presented to the user primarily through a **Virtual Data Catalogue**. A minimal implementation of this service would describe an application, its configuration and a list of input datasets. The existence of a service which delivers a concrete instance of a virtual dataset is assumed. The concrete instance could be obtained by locating a replica or by creating a new replica.

A **Data Selection Catalogue** will hold full the provenance chains of the virtual datasets, as well as any further information for selection purposes. Concrete replicas are recorded in the **Data Replica Catalogue**. The purpose of the **Job Catalogue** is obvious.

## Use Cases

The use cases are divided into categories.

### Data Acquisition

- 1.1 **Begin new run** - create new virtual dataset to reference written data.
- 1.2 **Acquire data**
- 1.3 **End run** - close open files and write Selection Catalogue attributes.
- 1.4 **Check RAW data file** - run application on single file dataset.
- 1.5 **Check RAW files** - merge files and run application.

### Reconstruction

- 2.1 **Virtual reconstruction** - create entry in Virtual Data Catalogue with RAW data and application/configuration information.
- 2.2 **Concrete reconstruction**
- 2.3 **Reconstruct file** - i.e. without waiting for the end of the run.
- 2.4 **Virtual production of Analysis-Oriented Data**

### Combining Runs

- 3.1 **Combining reconstructed datasets** - combine multiple runs into single datasets.

### Event Selection

- 4.1 **Virtual event selection** - i.e. catalogue input datasets and selection criteria.
- 4.2 **Concrete event selection**
- 4.3 **Separate into streams** - re-order the events in the dataset according to event characteristics.
- 4.4 **Copy selected events into new files**

### Analysis

- 5.1 **Analysis job**
- 5.2 **Select a task** - select an application configuration.
- 5.3 **Select an input dataset**

## Required Catalogues

From these use cases, it is inferred that the following catalogues are required:

- Application repository
- Task repository (a configuration for an application is called a *task* in AJDL jargon)
- Task selection catalogue
- Dataset repository
- Dataset replica catalogue
- Dataset selection catalogue
- Single-file dataset catalogue
- Virtual data catalogue
- Job catalogue

## BaBar Analysis Grid Application

### Introduction

The *BaBar Analysis Grid Application Use Case and Requirements* document dates from 2001 and describes a use cases and requirements for a basic Grid-based HEP analysis system. The use cases presented are briefly summarised here. The requirements are, for the most part, derived from the use cases and are therefore not repeated here.

### Assumptions

The application will be GUI-based. It will allow files in Root\* format to be located using a metadata query. Replicas will be distributed across different sites and the most efficient replica will be selected. Jobs will always be run where the data resides (in the early stages described here, at least) so no remote access to data will be necessary.

### Use Cases

#### Use Case 1 – Standard Physicist Analysis

- 1.1 The user authenticates once at his local site. The system does not differentiate between users.
- 1.2 The user prepares a binary version of his analysis code.
- 1.3 The GUI front end is invoked.
- 1.4 A unique ID is allocated to the user to identify his analysis run later.
- 1.5 A metadata query is used to identify the required input (logical) files.
- 1.6 Corresponding physical files are located and the best selected.
- 1.7 A set of script files describing the selected physical files is produced.
- 1.8 A configuration file is produced, describing the job configuration for each data file.
- 1.9 The user tells the GUI to submit the jobs.

---

\* BaBar use both Root format flat files and the Objectivity database for data storage.

- 1.10 The jobs are run on the nodes that hold the physical data files.
- 1.11 The system transfers the binary program and configuration file to the node that is to run the job and begins the run.
- 1.12 The node accesses the data on the local file system.
- 1.13 Any additional configuration files required are available by AFS.
- 1.14 When a job terminates, the output is copied to a user specified location.
- 1.15 When all jobs are complete any output concatenation required is performed.
- 1.16 The final output is copied to a user specified location.
- 1.17 The GUI informs the user that the analysis run is complete.

## **Use Case 2 – User Status Monitoring**

The user logs into the GUI and identifies an analysis run using a unique ID. The GUI gives the status of all the constituent jobs, and offers the opportunity to remove or re-submit failed jobs. Log files for the jobs are available.

## **Use Case 3 – Data Import**

Data is selected, as in U1.5. Taking into account the status of the data at the source (on disk, on tape, deleted etc), the amount of disk space available at the destination and the type of data being imported, the data are transferred to local storage. Different protocols are used for the transfer depending on the type of data being copied.

## **Use Case 4 – Repeat Analysis**

The user alters the program binary associated with an analysis run, and re-submits all jobs.

## **Use Case 5 – Site Problems**

The user notices that jobs sent to a particular site are failing (the basic system is not capable of identifying this problem) and re-submits all failed jobs, specifying that the problem site is not to be used.

## **Further Use Cases**

The document identifies further use cases that it would be helpful to write down:

- Monte Carlo production;
- Analysis using data stored in Objectivity; and
- Analysis of ntuples produced by Use Case 1.