BRAHMS Analysis Stages and event model.

F.Videbaek

July 13, 1998

(Draft)

This document will deal with the data model, data storage, as it is perceived now. The description of the user modules, details of data structures will be available in other BRAHMS software notes later.

BRAHMS Analysis Stages

Data occurs from many stages. Due to efficiency in accessing and the desire and need not to duplicate data, they will have exist in different files, part of the data possibly (event tags or run tags) in an Obj-y database. This will minimize duplication of data. Later analysis stages can thus proceed fairly efficient in so far the 'right' data are defined and kept for the later analysis stages in the same files.

The analysis stages envisioned to day are

- Simulated Event Generator Data e.g. Fritiof7.02, Venus.
- Generation of digitized data. This can happen in two distinct ways.
    1. GBRAHMS simulations will generate hit data from the event generator data using a detailed description of the detectors. This step will also generate digitized data in a slow or fast mode for the detectors.
    2. DAQ will generate raw data i.e. the data as will be written from the DAQ front end. It may be that another raw data format relating to the DAQ but created in simulations exists.
- Generation of Calibration database from raw or digitized data.
- Reconstruction, which logically includes generation, calibrated data (intermediate results) local tracks, tracks, and Particle ID, as well as global information
- Generation of Reconstructed Data Objects (RDO)
- Creation of Physics Data Sets (PhDS) [run, software & hardware trigger selection]
- Analysis of data using PhDS.

Each of these stages and the data they generated are discussed in more detail in the following sections.

Production of Event generator data

This includes running event generators like Fritiof, Venus, and others. At present the output format is Zebra based. It can be used in the GBRAHMS program, and in stand-alone programs for looking at expected physics distributions. It is worth to remember that the open model standard proposed by Y.Pang keep the event generator output as ASCII formatted files. We could change our Geant and analysis model to be able to read this. Maybe others (Longacre) have already done this for STAR? It is though put a low priority since a working scheme does exist.

GBRAHMS simulations

This stage takes the data from the event generators and creates events with hits and track information for active detectors. The data are generated from GBRAHMS and are presently written as c-stream files (flat files). The underlying data structures are table like i.e. a collection of structures. This step is quit time consuming using 1-4 minutes per Au Au event depending very much on centrality and angle settings.

- gtracks : Track parameters
- ghits : Hits in detectors
- gvolume : Geant geometry information

DAQ and Online Stage

- Raw data from the DAQ which can be in 3 different forms
  - translated
  - un translated
  - pedestal or non-pedestal subtracted data.
- Raw data from digitization routines. These will be like raw data, but in addition there will be relations data structures correlating digitized data with simulated hits. These relation or associations should be in separate data structures.
- It may turn out to be useful to have a level 0 conversion from a compact raw data format to the first level of digitized data in e.g. ROOT structures. Thus the raw data format are different from the digitized information presented to the first analysis modules. In this case a special module must be called to convert raw data into digitized data. This would possible also help in having the raw data in a more compact format.

Data Calibration

Calibration data (for the calibration database) first have to be generated. Hopefully much of this can be accomplished in near-real time. The second best choice is to have calibration performed

during the first step of reconstruction by first executing the calibration pass on the raw data files read from HPSS storage.

Data reconstruction

The first step is Raw-> calibrated data. This requires access to a database with calibration constants, and might additionally include simple clustering of tracking hits. Depending on the CPU needs these data could be short-lived, or persistent. The choice may depend on the need to go back to raw data in subsequent analysis.

Event reconstruction is a second step of data reconstruction. This will involve local and global tracks from the spectrometers. Most of the particle identification should be performed in this stage. It is not obvious that it can be completed. E.g. it is necessary to have the knowledge of tracks to obtain the 'final' time calibration of TOF walls before PID. At the minimum at first order calibration and PID should be archived in this stage.

The data objects stored persistently comes in several classes.

1. Hits on tracking detectors; track to hit associations.
2. Local tracks in tracking stations
3. Global spectrometer tracks
4. PID information
5. Multiplicity information, vertex information
6. Physics tracks

In the MDC these will be on a single output file (root file). In the production mode we will like to split these to produce RDO files from which Physics Data Sets can be extracted efficiently by run and trigger selection, and the more voluminous output of hits, associations, calibrated data. In production mode it is likely that these will only have to be stored for a smaller subset of data (10-15%).

Certainly early on but also later it may be needed to re-run part of reconstruction to get proper particle identification. Another possibility may be depending on i/o versus cpu needs to have standard modules included in the subsequent PhDs generation.

Physics Data Set generation

This data pass will select from the large sample of Reconstructed Data Objects those that can be used in specific physics analyses. Such sets can e.g. consists of

- Specific particle kinds and trigger selection for many angle settings
- Detailed global information (multiplicity, beam-beam counter).
- Two-particle data from a large number of data files, but only a single angle setting.

- Calculation of tight multiplicity cuts to select out very central events.

These runs can be selected based on the information in the run database as well as from the event header summary (event tag file, run summary on disk). These selection tasks have to be coordinated to get reasonable access to the HPSS file system and transfer speed. This will most likely be done using the ROOT frame work; maybe using a PROOF like system if the same files are used extensively by multiple analysis/ data set generations.

Analysis of Physics Data Sets.

Some of this analysis will be done at RCF, but a significant fraction will be done at the collaborators home institutions.

Overview of data objects needed

In addition the division on data on the stages (passes), it seems reasonable also to think on a division of data on the level of detector parts. Each of these parts may or may not be on the same kind of storage e.g. data from TPCs could be on different files. This would help in speeding up later analysis and selection.

Event tags

They are short data structures that includes references (keys) and very truncated information on the other pieces of data belonging to events. The division below is most relevant to raw and calibrated data. The main reason to have these is that the event object model must be aware of data being distributed of several physical files. It can thus not easily employ a comprehensive object model as is done in many ROOT examples. They rely on using pointers as well as memory or disk access to all objects.

Global Data

- Beam-Beam counters
- Multiplicity
- ZDC
- DAQ, trigger information

Mid Rapidity Data

- MT1, MT2
- TOFW
- WM1, WM2

Forward spectrometer (FS)

FFS
T1, T2
WF1, WF2
H1, C1

BFS T3, T4, T5
H2, RICH

For later stages it may be more natural with three divisions according to detector component.

Global data

- Centrality selection
- ...

MRS

- Tracks

FS

- Tracks, PID information

This part of the doc has not been re-written (6/28)
The BRAT implementation is partly based on some of the concepts.

Data structures and analysis modules.

The data structures should be as independent as possible from the analysis modules, which serves as control modules.

The data structures to be maintained by the analysis code can be of many different kinds. The ones being considered so far are

Tables

Tables are a simple kind of structures. They can be dealt with by having a table header and a list of table objects. It is necessary for the table header to have a description of the data in order to store it onto persistent storage (flat files, Obj-y databases, ROOT file).In STAF the table components are defined using the Corba interface definition languages (IDL files). Obj-y requires .ddl files while ROOT uses the class description from the .h files.

Generalized event objects

This could be a list of other event objects or a list of some fundamental objects (like TPC clusters, etc). For these to be stored the same kind of argument holds. Thus a description is needed. ROOT does this by defining the objects with the classdef macro's. The files contain the name and version of an object. The code necessary to read and manipulate the objects must be linked into the reading program e.g. using shareable libraries. The objects can only be dealt with if the code reading these can defined them dynamically, and has a reading algorithm present. In other methods e.g. when using an object oriented database like Objectivity all the data objects are derived from a base persistent class, and defined special by description files (DDL files). It is a concern if the data model changes i.e. schema generation and schema evolution.

User modules

These are the control modules that has a set of well defined entries such that they could be callable by a simple framework or command language. It must be possible to transfer the knowledge of data from the general frame work to user modules without being specific. As an example expressed using C++ like code

```
User->event( evtobj* obj1 const, evtobj* obj2);
```

Note the separation the input obj1 and output obj2; the routines are allowed to add to the obj2, but not to obj1. It might even be very useful if all objects are named such that a module can specify what objects are required on input and expected on output. Within the object model described above that is based on ROOT this is achieved naturally.

Thus the framework deals with two quite separate entities, namely event objects and modules which can be called at different instances

```
module->Begin(Job*)[1]
module->Event(Evtobj,...
module->Finish(...)
```

Other issues:

To keep track of what data which belongs to the same event when multiple structures and files are in use is not a trivial task. It can be done by defining trees or directory structures that contains the information on the event and run number. The event objects as defined above could be the ROOT object for a tree which belongs to a given analysis stage (raw, dst etc.).

- The use of detectors vs. modules;

I would think of a detector more as an object, which has some 'geometry' operators associated with it, rather than performing the 'actions' on hits as you envision. I would tend to use the term Analysis Module more. An analysis module may or may not refer to the 'geometric' properties of a detector. This train of thought come from considering what happens in the later analysis stages e.g. combining tracks, matching tracks with TOF hits, where the object of interest is not a detector.

There is certainly use for a detector concept e.g.

[1] At present the Begin() and Init() methods do not have an object in the call.

detector

- geometry, coordinate transformation, display, digitization parameters
- display of hits, projecting tracks to detector etc

Some module may need to know about detectors. This may lead to some kind of object division in classes

-event objects
-geometry objects
-analysis modules

Objects in these should be allocable dynamically and searchable by name. The default Event() entry are called with an event object as input and one for output.