LEP Data Archive

http://wwwinfo.cern.ch/pl/cernlib/rd45/references.htm

A project has recently been proposed whereby existing LEP data would be archived for a period of some 30 years. Over such a long period of time, it is assumed that little of today's computing environment would remain. In particular, it has been assumed that the current CERN Program Library, existing operating systems and the Fortran programming language will not longer exist. It is estimated that a few TB of data per experiment will need to be converted and stored, giving a total data volume of perhaps 20TB. It is currently assumed that the data will be stored in Objectivity/DB and that a demonstration of an analysis against the database made.

GOALS, ACTIVITIES, PROBLEMS

Goals:

- save DELPHI data for future generations of physicists;
- provide a framework for moving towards OOP and ODBMS;
- SW development experience for future experiments.

Activities:

- save selected datasets into Objectivity/DB;
- provide a library for accessing events;
- provide an example of physics analysis code.

- OO and ADT paradigms;
- mixture of C++ and FORTRAN;
- Objectivity/DB as available ODBMS.

Problems:

- Data Model;
- Input/Output System.

OOP, C++, ODBMS

OOP

- known for more than 30 years;
- the term is frequently misused, often ADT is assumed; OOP is to express commonality in class behaviour;
- does nothing to remove the complexity of the program.

+ Allows to create components.

? Is amount of comonality big enough ? Is there a scope for more than data abstraction ?

Hybrid design – apply variety of styles as need dictates.

C++

+ supports data abstraction;

+ supports OOP;

– difficult to learn, use, debug;

– less efficient than FORTRAN.

Mixing C++ code with code written in FORTRAN.

ODBMS

+ higher level of abstraction than that of RDBMS or ZEBRA;

– disk space;

? efficiency; distributed environment.

What Is Objectivity/DB ?

"Objectivity/DB is a distributed client/server object database management system (ODBMS) for applications that require flexible information modeling, involve complex relationships, and demand high performance."

Logical Storage Classes:

- A basic object is the fundamental unit of storage.
- A container is a collection of basic objects.
- A database is a collection of containers.
- A federated database logically contains user-defined databases and the object model (or schema) that describes all publicly visible class definitions. The schema is created using the Data Definition Language (DDL) processor.

DATA MODEL

- Datasets
  Events are organised by collections (datasets).
- Run associated info:
  - beam energy;
  - beam spot;
  - run quality.
- Event:
  - EventId
  - Pilot info
  - Info related to whole event, eg b-tagging
  - Verticies
  - Tracks
  - Detectors info for tracks
  - Hits $\leftrightarrow$ Tracks mapping
  - Verticies $\leftrightarrow$ Tracks mapping
  - Detectors info $\leftrightarrow$ Tracks mapping
- MCEvent inherits from RDEvent and adds analogous information from simulation

DDL SCHEMA

Two Federated Databases – LEP1 and LEP2.

Databases: “Main”, “Run”, “Datatags”.

“Main” database:

- “Dataset” container contains general information about the dataset.
- “RunInfoCont” contains run related information.
- “EventIdCont” contains EventId’s for events.

“Run” databases (Event related containers):

- EventCont – general information about event
- VertexCont – verticies
- TrackCont – tracks
- Detector – detector information related to the whole event
- TrkDetector – detector information related to the tracks

“Datatag” databases – similar to the current Eventlists.

DDL SCHEMA (cont)

```
class EventId : public ooObj {
    int32 runNumber;
    int32 eventNumber;
    ooRef(Event) itsEvent <-> itsId;
};

class Event : public ooObj {
public:
    ooRef(EventId) itsId <-> itsEvent;
    ooVArray(int32) itsHeader;
    ooRef(Vertex) itsVerticies[] <-> itsEvent;
    ooRef(Track) itsTracks[]     <-> itsEvent;
    ooRef(DetectorInfoList) itsDetectorInfoList;
};

class Track : public ooObj {
public:
    ooRef(Event)  itsEvent     <-> itsTracks[];
    ooRef(Vertex) itsVertex    <-> itsTracks[];
    ooRef(Vertex) itsEndVertex <-> itsIncomingTrack;

    float32 px, py, pz;
};
```

PHDST – DELPHI INPUT/OUTPUT SYSTEM (old)

```
*
* Stage the tape EX1037 and process all events.
*
MEDIUM = TAPE, MOUNT = STAGE, VID = EX1037
*
* Process only DST records from the file
*
FILE = /usr/delphi/work/mydata.dat, REC = DST
*
* Stage tapes from EA0001 to EA0020 and EA0033
*
MED=TAPE, MOUNT=STAGE,
          VID=(EA0001-20,EA0033)
*
* FATMEN name is used to define
* the set of tapes
*
FATMEN = DST092          ! select all data of 1992
FATMEN = DST092/C35      ! select one tape only
FATMEN = DST092/C1-10    ! select the first ten tapes
*
* Example of a medium for output.
*
VID=EX1234, STREAM = 1, COMPRESS = X
```

DELPHI INPUT/OUTPUT SYSTEM (new)

Design rules:

- User interface should be close to an existing I/O system, i.e. to PHDST.
- Simultaneous work both with Objy and old Zebra format.
- Support for multiple views: the pieces of events are divided into several parts, user can select desired pieces of events, the database can contain another objects than user sees.

Main classes visible to the user:

- delios – similar to the PHDST package; provides a loop over events;
- UserCode – a derivative from this class is provided by the user and contains a user code.

Example of possible user program:

```
#include <delios.h>
#include "MyCode.h"

int main()
{
    delios phdst ( delios::PDL, "myPDLfile" );

    UserCode * uscode = new MyCode;

    phdst.Run(uscode);
}

class MyCode : public UserCode {
    virtual void Init() { user00(); }
    virtual void Pilot() { user01(); }
    virtual void MainProcessing() { user02(); }
    virtual void Term() { user99(); }
};
```