Object Oriented Data Analysis in the DELPHI Experiment

T.Camporesi
P.Charpentier
M.Elsing
D.Liko
N.Neufeld
N.Smirnov
D.Wicke

GOALS

- provide a framework for moving towards OOP
- provide DELPHI data for future generations of physicists
- attract and maintain the interest in analysing the DELPHI data far beyond the end of the LEP data taking
- gain SW development and physics analysis experience with the new HEP programming technologies

PROJECT SCOPE

The project can be roughly divided on three parts:

- creation of object model of DELPHI data
- providing means of data access
- creation of data analysis tools

They are not independent $\Rightarrow$ design iterations

Histogramming package is outside the project scope

INFORMATION CLASSIFICATION

- **Datasets**
  - Events are organised by collections (datasets)
  - The datasets are subdivided in runs
  - Runs are spread over several files (runfiles)
- **Runfile associated info**
  - beam energy and position
  - luminosity
  - run quality
- **Event**
  - EventId (run, event number)
  - General info – summary of its physics characteristics
  - Info related to the whole event, eg trigger info
  - Vertices and Particles
  - Tracks and Clusters
  - Detector info for the tracks/clusters
  - Associations between different information elements, eg simulated $\leftrightarrow$ reconstructed

DATA MODELS (1)

Simplifications:

- only the result of the final processing stage (DST)
- only the last (the best) version

- **Persistent model** – how data stored
- **Transient model** – how user sees them

DATA MODELS (2)

Presently supported persistent models:

- Zebra as is.
- Objectivity/DB as OODB test:
  - DELPHI data information
  - user access patterns

Definition of the transient model:

- Analysis use cases
- Layering of info:
  - experiment independent
  - experiment dependent
  - DELPHI specialist dependent

DELIOS – DATA ACCESS PACKAGE

Design requirements:

- User interface should be close to the currently used I/O system
- Platform-independent specification of external media
- Support both Objy and Zebra format and conversion
- Events are divided into several parts, user can select desired pieces of events
- Persistent objects need not map one to one to the transient objects

DELIOS: Main classes

Passive "user interface": the loop over events is inside the package.

Main classes visible to the user:

- delios – this is a supervisor; provides a loop over events
- UserCode – a derivative from this class is provided by the user and contains a user code
- Event

Example: main()

```
class MyCode : public UserCode {
    virtual void Init();
    virtual void Main(Event &);
    virtual void Term();
};

int main()
{
    // Create an object 'dst_analysis' of class 'delios'.
    // Mode: sequential. I/O media are in the 'MY_DATA' file.

    delios * dst_analysis = delios::Get(delios::seq, "MY_DATA");

    // Create an instance of UserCode

    UserCode * uscode = new MyCode;

    // Run the system giving it UserCode

    dst_analysis->Run(uscode);
}
```

DELIOS: Media specification

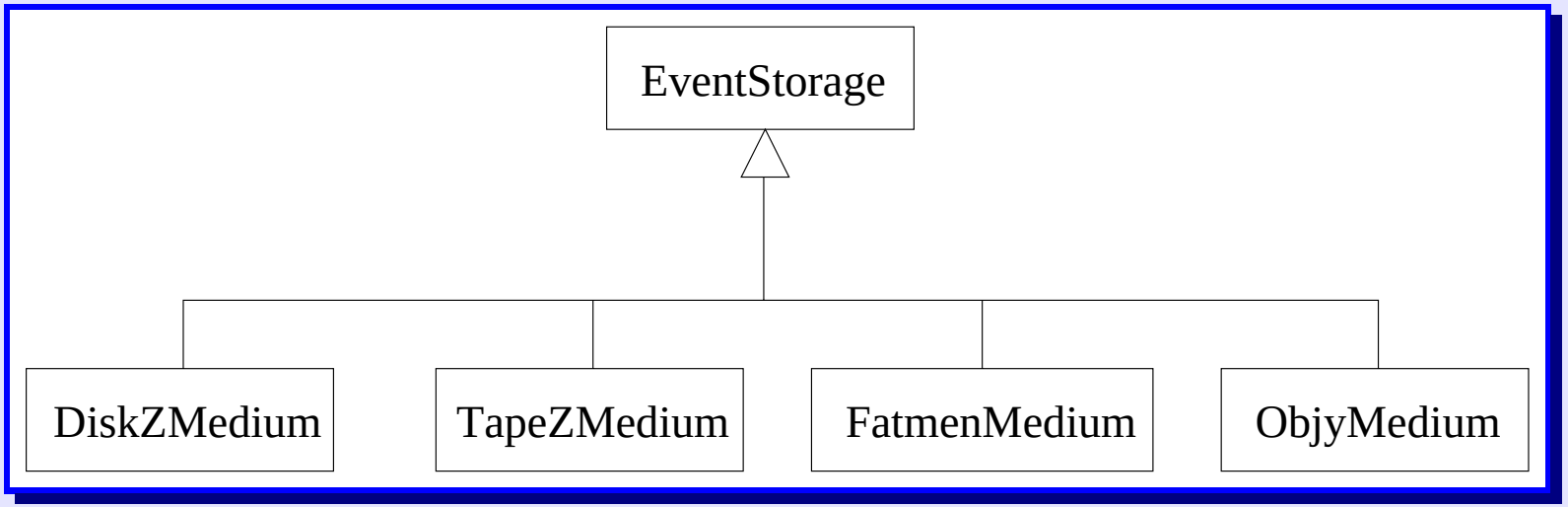
DELIOS allows the specification of external media in a form independent from the operating systems and operating environments.

```
* Contents of the MY_DATA file

medium = disk, name = /usr/nsmirnov/data/my.data, format = zebra
medium = tape, name = EX1000, format = zebra
medium = objy, name = lep1/dst94
```

DELIOS: Hiding the media implementation

- **all event storage classes inherit a common interface defined by the abstract class** EventStorage**.**



- **all classes dealing with the same kind of storage are put in separate dynamic loading libraries which are loaded at run time if needed.**
  - **libdelios_z.so works with CERNLIB**
  - **libdelios_o.so works with Objy**

DELIOS: Proxies

The package widely uses the proxy design pattern. For example, the real content of the event is in the class EventBody **hidden from the user to which the class** Event **visible for the user works as a "smart pointer".**

Some consequences:

- **the "object inflation" concept in accessing the event information. Parts of the event are loaded into the** EventBody **from the external medium only if the user is demanding access to them.**
- **allows reference counting of objects.**

DELIOS: example

```
void MyCode::Main ( Event & theEvent )
{
    vector<int> header;
    theEvent.GetHeader(header);
    if ( header.size() == 0 ) return;

    int nevt = header[HEADER_NEVT];
    if ( nevt > 100 )
         throw EndOfUserJobException();

    // Access to general information

    int vers = theEvent->version;
    cout << "Reconstruction program version "
         << vers << endl;

    // Calculate number of charged tracks

    int ntr = 0;
    Event::VertexIterator iv;
    Vertex::TrackIterator it;
    iv = theEvent.GetVertexIterator();
    while ( iv.Next() ) {
      it = (*iv).GetTrackIterator();
      while ( it.Next() )
        if ( (*it)->charge != 0 ) ntr++;
    }
}
```

EVENT ANALYSIS TOOLS (1)

Event is mainly a tree structure of alternating vertices and particles.

ParticleExtractor – the extraction of particles from the tree into a list is a key operation. It is based on the properties of particles (charge, particle id, quality cuts etc) and vertices (V^0, kinks, γ-conversion etc).

ParticleSelector – having selected lists of particles it will be possible to further reduce the contained particles. Operates on information of an individual particle without using information on the structure of the vertex-particle tree.

Particle List Tools: thrust, jet clustering, multiplicity, etc.

Event Tools: There are also quantities that cannot be calculated on basis of a particle list. These will take the complete events structure as their input. Example: b-tagging.

EVENT ANALYSIS TOOLS (2)

We want to follow in the development an iterative approach.

In order to verify the design developed so far the use case of a QCD event shape analysis will be studied.

- Relative simplicity – it does not require detector specific information and association of particles to their generator information.
- Several physical observables, as the event thrust, or methods, as acceptance correction, are contained also in the use cases of many other more complicated physics analyses.

In the next iteration the design will be improved profiting from the experience obtained in this simple case. In particular access to the simulation information and detector information will be implemented.

SUMMARY

- persistent and transient object models of data
  - layering of info
- means of data access
  - platform-independent specification of media
  - hiding the media implementation through abstract class and DLL
  - proxies
- data analysis tools
  - under construction

Prototype is expected for summer