Object Oriented Data Analysis in the DELPHI Experiment

T.Camporesi[1], *P.Charpentier*[1], *M.Elsing*[1], *D.Liko*[2], *N.Neufeld*[2], *N.Smirnov*[3,1], *D.Wicke*[1]

[1] CERN, Geneva, Switzerland
[2] Institut für Hochenergiephysik Österreichische Akademie der Wissenschaften
[3] Institute for High Energy Physics, Protvino, Russia

Abstract

The paper presents the current status of the project for building an object-oriented programming environment in the DELPHI experiment. The main project goal is to make possible the analysis of the DELPHI data far beyond the end of the LEP data taking.

Keywords: data analysis, OOP, C++, DELPHI

1 Introduction

Object Oriented Programming is the key technology for the next generation of HEP software. An interest of the DELPHI collaboration members in this technology as well as the need of archiving the DELPHI data for the future generations of physicists, were the main reasons for creating a project for development of an Object Oriented analysis framework in the DELPHI experiment.

The aims of the project were formulated in [1] and they are repeated here:

- to prepare the saving and archiving of DELPHI data for a long period of time over which it is assumed that today's computing environment would no longer remain;
- to provide a basis for moving toward Object Oriented Programming;
- to attract and maintain the interest in analysing the DELPHI data far beyond the end of the LEP data taking;
- to gain software development and physics analysis experience with the new HEP programming technologies.

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.

These parts are not independent, for example, modifications in the design of the analysis tools can affect the data model.

The data analysis tools aim on the evaluation of physics quantities, ranging from event shapes to secondary vertices. For interactive analysis and visualisation the project relies on available tools, either PAW, ROOT [2] or a future LHC++ analysis tool.

2 Data Model

The creation of an object model of DELPHI data is the first indispensable step in the development of an Object Oriented analysis framework.

The DELPHI experiment had two big data-taking periods: LEP1 and LEP2. The data from both periods are organised in collections called datasets. Each dataset contains the events of one year of LEP data taking and corresponds to a given software reprocessing. Events are reconstructed several times, which reflects the improvement in the understanding of the detector performance and physics goals. Only the last and presumably the best version was considered in the object model.

The datasets are subdivided in LEP fills which in turn are subdivided in runs each one covering a data taking time with stable LEP and detector conditions. The runs are attributed run numbers as determined from the online system. They are spread over several files with lengths optimized for the output media and further manipulations. A runfile is the smallest event collection unit of DELPHI. The information associated with the LEP running conditions as the beam energy and position (spot), detector quality, luminosity etc. is organised on the runfile basis.

DELPHI events consist of :

- Pilot information – Event Identifier (run, event number) and a summary of its topological and global physics characteristics;
- Vertices and Particles;
- Detector information for the particles;
- Information related to the whole event, e.g. b-tagging, unassociated hits;
- Associations between different information elements, e.g. vertices – particles.

Monte Carlo data can be classified similarly with some complications related to the MC generator parameters.

Typically there are two main patterns of event access: (1) sequential when the user needs almost all events from the dataset and (2) random when the user needs a small subset of selected events. More detailed analysis of the user access patterns depends on the underlying implementation of data store (ZEBRA or Objectivity/DB or whatelse). The final choice of the data store has not been done yet.

3 Data Access Package

The second part aims at the development of new object oriented tools for the data access necessary for physics application. The following design requirements were considered when building-up the data access package:

- User interface should be close to the currently used I/O system;
- Platform-independent specification of external media;
- Simultaneous work both with Objectivity/DB and old ZEBRA format;
- Events are divided into several parts, user can select desired pieces of events;
- The datastore can contain another objects than user sees;
- The package must support multiple views of the same events.

The package implements a passive “user interface” which hides the loop over events inside the package and all interactions with the user are done via set of interface functions (“callbacks”). These interface functions collected together in the abstract class `UserCode`. The physicist performing an analysis implements the virtual methods relevant for the data analysis providing an own class which inherits from `UserCode`.

The peripheral devices (files, tapes etc.) needed for physical applications are defined in DELPHI by a simple language used for many years. It allows the specification of external media in a form independent from the operating systems and operating environments at CERN and extern laboratories. This approach proved to be successful and it has been preserved with some extentions in the new package. The medium specification syntax is very simple and can be easily understood by following the example:

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

This example contains several medium specifications. Each one identifies the medium by defining values to a set of standard keywords (medium, name, format etc.). For example in the second specification the medium is defined as a tape with a given VID name (EX1000) and data in ZEBRA format. The third specification defines the `dst94` data set of Federated Data Base `LEP1` in Objectivity/DB.

Important design goal was to make the main part of the system independent from the particular implementation details of the real media. This goal is achieved by two ways. First, all event storage classes inherit a common interface defined by the abstract class `EventStorage`. Second, all classes dealing with the same kind of storage are put in separate dynamic loading libraries which are loaded at run time if needed. If for example the user specifies only disk ZEBRA files then the library working with ZEBRA will be loaded and the library working with Objectivity/DB will not be loaded (it even can be absent).

The package widely uses the proxy design pattern [3]. The idiom handle/body is used to designate the proxy and the reference object. For example, the real content of the event is in the hidden from the user class `EventBody` to which the visible for the user class `Event` works as a "smart pointer". This allowed to implement the "object inflation" concept in accessing the event information. Parts of the event are loaded into the `EventBody` from the external medium if only the user is demanding access to them. This idiom also allowed to implement reference counting of objects.

Trivial but important feature of the package is absence of physics algorithms, that is, the package provides means of event access and only such means. It contains no algorithms which can modify the event or calculate physics parameters of the event. Such algorithms belong to the event analysis tools and put in a separate library.

4 Event Analysis Tools

In order to enable future generations of physicists to successfully analyse DELPHI data it is necessary to have a clean and easy to use analysis tool. As much of the experts knowledge that is currently passed from student to student in form of FORTRAN code should be moved into publically available tools. Especially state of the art event selection from todays experts should be provided as starting points for the future user, without sacrificing the possibility to investigate new algorithms. Special care needs to be taken to allow easy calculation of the most often used quantities like e.g. acceptance correction. Finally an extensive documentation will be of crucial importance.

After a first analysis of these requirements and a long list of further requirements an initial design concept was developed. The basic design idea is to create classes which can be used to perform the most complicated and advanced analysis and to derive specialised classes from these to present the user simplified interfaces for standard tasks. The specialised classes in this scenario later serve as important examples for the documentation of advanced features.

The event stored for DELPHI as described above distinguishes vertices, particles and detector information. A tree structure of alternating vertices and particles allows to store the relation ship between reconstructed particles, kinks, etc. and their measured decay products, as well as the complete decay chain of simulated particles. Detector information is connected to each particle.

This structure will remain the basic structure for DELPHI data, though the C++ interface will hide duplicate and obsolete information. As most physics quantities deal with lists of particles, the extraction of particles from the tree into an list is a key operation. The particle type (charged, neutral, id), the vertex type (reconstructed V0, kink, photon conversion, etc.) will play the key role in this process. In addition elaborated analysis may want to define quality cuts to decide whether to accept certain particle or vertex types. The interface needed to fulfil these requirements will be developed during the implementation of use cases.

Having selected lists of particles it will be possible to further reduce the containing particles through a particle selector. The particle selector itself thus is a class that can operate on information of an individual particle without using information on the structure of the vertex-particle tree. The particle list will serve as input to many calculations. These calculations are summarised under the roof of particle list tools. The particle list tools will be a loosely connected set of classes which provide the algorithms using identical or similar interfaces. Typical examples of particle list tools are thrust, jet clustering, multiplicity, etc. There are also quantities that cannot be calculated on basis of a particle list. These will take the complete events structure as their input and thus are called event tools. The main example of an event tool will be the event b-tag.

In the development we want to follow an iterative approach. In order to verify the design developed so far the use case of a QCD event shape analysis will be studied. The current implementation of the framework will be used to develop a corresponding QCD event shape analysis program.

The event shape analysis was chosen because of the relative simplicity of the task. It does not require detector specific information nor association of particles to the generator information. In addition 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 associations of particles to their simulation information and access to detector information will be implemented.

References

1 N.Smirnov, F.Carena and Tz.Spassoff, “Data Archiving in the DELPHI Experiment”, CHEP’98, Chicago, Autumn 1998.

2 `http://root.cern.ch`

3 E.Gamma, et al. “Design Patterns”. Addison-Wesley. 1994. ISBN 0-201-63361-2.