**DELPHI Collaboration**

DELPHI 2000-151 PROG 241
20 July, 2000

# Ensuring long time access to DELPHI data: The IDEA project

**Tiziano Camporesi, Philippe Charpentier, Markus Elsing, Dietrich Liko, Niko Neufeld, Nikolai Smirnov, and Daniel Wicke**
CERN

**Abstract**

The long term accessibility of its data is an important concern of the DELPHI collaboration. It is our assumption that the storage of the data by itself will be a minor issue due to the progress in storage technologies. Therefore DELPHI focuses on a reorganisation of the data, which should provide a flexible and coherent framework for physics analysis in the future. A project has been started, IDEA (Improved DELPHI Analysis), to provide an implementation based on modern software technology. It will ensure usability for future generations of physicists.

# 1 Introduction

Scientific ethics demands the preservation of data collected by the four LEP experiments. Given the amount of data and the decreasing costs of storage media, the DELPHI collaboration sees the challenges not in storing the data but in retrieving and interpreting these. This note presents the effort to address this problems by developing a legacy free analysis framework.

We are conscious of the fact that the computing environment in high energy physics is currently changing. Modern software technologies are being introduced and there is a shift of the computing paradigm. We expect that at the end of the running of the LEP experiments also physicists still engaged in the experiment will orient themselves towards the new technologies. In addition a new generation of physicists, who without doubt will be committed to the new experiments, might be deprived from access to our data due to a "software generation gap".

The aim of our project is therefore to develop a new analysis framework for the DELPHI environment in object-oriented style. The programming language of choice is `C++`, which offers not only the necessary features, but has also a proven record of industrial strength software. We plan to keep the project simple and independent from other software technologies. This should ensure long time usability even in a changing and still evolving environment.

We expect that this tool will allow physicists in the future to perform analysis of DELPHI data, without the detailed knowledge necessary today. Our emphasis is therefore to develop a framework, that is easy to use for physicists, who do not have access to this information. We foresee that the physicist in the future will perform some re-analysis of the data either due to new ideas from theory or due to new experimental findings. On the other hand we think it will be difficult to repeat high precision analyses in the future, as the necessary detector expertise will have been lost.

We aim to have this new software environment commissioned before Spring 2001 in order to have it actively used by people, who have also expertise in our standard analysis package. We are convinced that this active field test is essential to ensure a software package well debugged. Our goal is to have several physics papers produced using this new software by the end of 2001. This package is developed as a competitive alternative to our standard analysis package (SKELANA [1]), not as a replacement. We plan to keep our FORTRAN based solution in parallel at least until 2005.

# 2 Problem specification and requirements

The new analysis framework should allow future generations of physicists to work with data without the need to become a "DELPHI expert". One basic assumption was that the data are derived from the final processing and no additional fine tuning will be necessary. This remedies the need to provide numerous detector specific packages, that are currently an important part of our analysis programs.

All information available for a given run or event should be accessible via the same interface, thereby removing any need to manually access additional data contained in separate files or data bases. The data should be accessible in a intuitive and abstract form. Ambiguous information should be removed and obsolete informations should be

hidden.

A detailed analysis of the existing software and data structures has been undertaken to understand the needs for meeting these goals.

## 2.1 Use Case Analysis

The most important consideration for us is the usability and acceptance by physicists. We concentrated therefore on the analysis of so-called "use-cases" [2]. In our case these use-cases are exemplary physics analyses, which should be supported by our framework.

A quite general category of analyses can be summarised as *QCD use-case*. Typical for this kind of analysis is that well defined algorithms, such as the calculation of event shapes or the reconstruction of jets, are applied to the event data. These algorithms typically operate on a selection of particles. It has to be possible to apply these algorithms not only on reconstructed quantities, but also on simulation without detector effects. Some quantities, that are derived from event data, have to be provided, such as the effective centre of mass energy of a high energy event ($\sqrt{s'}$). In most cases access to more detailed detector information is not necessary.

Other requirements were derived from a *Heavy flavour use-case*. As a typical example we studied the reconstruction of a $D^*$ signal. This added the need to access trajectory information and association between reconstructed and simulated particles. It is also necessary to provide tools for secondary vertex reconstruction and kinematical fits. Another important tool is the beauty quark tagging.

Finally we have also studied a *LEP II search use-case*. We found that most tasks can be performed with tools already identified before. Some additional tools, as constrained fits for jet systems are necessary.

It is our aim to develop a framework, such that these analysis can be performed in a simple way. In addition to the framework we plan also to provide reference implementations for each use-case.

# 3 Design

As a result of the use-case analysis, entities can be categorised in two classes. One class is concerned with the representation of data recorded by the experiment. The other class concerns entities that represent physics algorithms and their implementation. These elements provide the user - the physicist - with a flexible analysis environment. For a specific problem these elements can be assembled in an easy way to form the program.

## 3.1 Experimental data

Two different kind of experimental data could be identified. One is the actual data recorded for an event, the other is additional information, that is the result of monitoring and calibrating the detector. Both types of data are provided to the physicist in a homogeneous environment by means of the DELIOS package [3]. As an additional advantage this package decouples the data from the underlying storage medium. Currently this allows to access event data stored either in ZEBRA banks [4] or Objectivity databases [5]. In the future it could be adapted to the technology of choice. In the following the organization of the event data is discussed in some more detail.

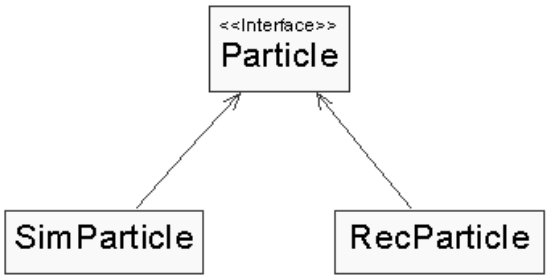


Figure 1: The inheritance relation used to model the dependence of `RecParticle` and `SimParticle` to their common interface `Particle` is shown by an UML diagram [7].

### 3.1.1 Organization in layers

A fundamental aspect is the organization of the event data in a layered model. On the highest abstraction level the event data are organized as entities that are close to our understanding of collision events. Events are seen as cascades of decaying particles. This leads naturally to an organization of the data in trees consisting of particles and vertices. On this level event data can be seen independent from the actual experiment.

Complementary to the reconstruction information also simulation information is provided on this level. There are two different kind of information, one concerning the underlying event generator, the other concerning the detector simulation. The information is again organized naturally as a tree of particle and vertices. For some analyses this level is already providing enough information.

On the second level the data is seen as trajectories, reconstructed by the tracking detectors, and showers seen in the calorimeters. While this abstraction is already closer to the actual event reconstruction, it still can be seen in a way independent from the experiment. Again there is a class of analyses that will need no more information.

On the third level the data is seen as actual detector data. This information is specific to the DELPHI experiment. Its level of detail corresponds to the DELPHI short DST [6].

#### The Particle and Vertex layer

Reflecting the fact that similar information is provided by reconstruction and simulation, a pattern based on interfaces has been chosen. Entities are defined by a pure abstract base class *Object*. For simulation information a `Sim`*Object* is derived, that implements all attributes and values. For reconstructed entities or quantities a `Rec`*Object* is derived that additionally provides information on the measurement uncertainty, if appropriate.

This strategy allows on the one hand to use *Objects* to implement algorithms, which can be applied to simulation and to reconstruction information. On the other hand algorithms can be implemented in terms of `Rec`*Objects*, in case the full reconstruction information is necessary. This scheme has been applied to entities like the particle itself (`Particle`, `RecParticle` and `SimParticle`, see figure 1), but also to quantities like the particle charge.

A second design choice was to express the tree structure of particles and vertices by a tree container, that has been implemented similar to the standard containers provided by the STL.This separation of objects and their associations provides additional flexibility.

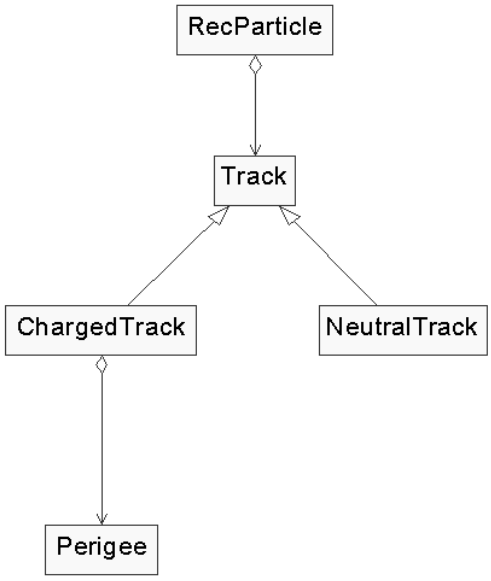


Figure 2: Design fragment of the track layer shown by an UML diagram. `RecParticles` can have associated `Track` information. Depending on the type of the track an appropriate `Track` object is chosen, currently a charged or a neutral track. Detailed geometrical information are the member attributes of these objects.

A real data event contains one particle vertex tree, `RecVPTree`. A simulated data event has in addition a second tree, `SimVPTree`. In a similar way the relation between simulated and reconstructed entities is not implemented by links but by external associators based on STL multimaps. Again this increases flexibility, as the choice of association between simulation and reconstruction data might not be unique.

It was found necessary for many applications to deal with selections of particles, *particle lists*. To allow easy exchange, they are commonly expressed as list of particles stored in a STL container `vector<RecParticle*>`.

### The Track layer

This layer is concerned with the true geometrical shape of the particle trajectories. A particle can be associated to a track, which contains information that are as close as possible to the original measurement (see figure 2). Currently we have concentrated on the design of the trajectory of charged tracks. We plan to complement this layer not only with information on neutral showers but also with detailed information on reconstructed decays, as so-called $V^0$s and hadronic interactions.

The most important aspect of our strategy is the use of the information closely connected to the actual measurement. For charged tracks in DELPHI this implies trajectory information in so-called perigee parameters and their uncertainties [8]. Other representations, as for example the four-vectors in the particle layer, are derived from this information. Basic functionality, as for example the extrapolation of the perigee to arbitrary vertex positions, are provided as methods and allow to profit from the precise information in an easy way.

This layer is essential for many physics analysis that depend on precise geometrical information, such as the calculation of impact parameters, secondary vertices or invariant masses.

Together with the information in this layer we have also developed tools that are necessary for physics analysis, like an object oriented version of our vertex reconstruction.

### 3.2 The algorithm objects

Physics analysis are sharing a number of algorithms, that operate on event data to provide useful information. Typically a physics analysis would create several instances of such algorithm objects and evaluate their results in turn.

Some of these algorithms need access to all event data, the so-called `EventTools`. An example is the calculation of the effective centre of mass energy at high energy events or the beauty-quark tag. Other algorithms are defined naturally on subsets of the particles, on particle lists, and are called `ParticleListTools`. A typical example is the evaluation of event shapes and the reconstruction of jets.

Common to all algorithm objects is a `calculate_on()` method. Corresponding to the algorithm object this method takes the event or a particle list as an argument. On evaluation the algorithm object remembers the result and is providing it to the physicist by specific access methods.

Because similar information might be obtained by different implementations, we chose to provide abstract base classes to represent the result of algorithms. This allows to pass the result of one algorithm to another, without introducing a dependency on the specific implementation.

This scheme is general enough and it can be adapted to other kind of tools that are anticipated to turn up in the further development of IDEA. It is also assumed that this scheme is adopted by the users to provide more and more "ready to use" particle list tools, so that other colleagues may profit from their work.

## 4 Status

The framework has been implemented to the extend of the *QCD* and the *Heavy Flavour use case*. The particle and vertex layer as well as the track layer are implemented as specified by the design. A number of event and particle list tools have already been released. Few more complex tools are currently available by an implementation that provides an object oriented wrapper to the legacy implementation.

After a review of our design, we plan to complement the features by working on the next use case. Our next aim is to design and implement the access to detector specific information.

We see the use of our framework by physicists, who might not be `C++` experts, as an important benchmark of the usability. Thus we plan to reach by autumn a phase where we can start first "field trials".

# 5 Conclusions

We believe that the data from LEP should be accessible much beyond the end of the LEP collaboration themselves. Therefore we consider it our duty to provide a software environment allowing access to DELPHI data which should be working at least until the end of the exploitation of the LHC experiments.

We chose to develop the IDEA package to access our DSTs based on object oriented technology and implemented in `C++`. IDEA will allow user friendly access to the DELPHI data to the new generation of students and to the people who are willing to use modern computing tools in the short to medium term. In the long term, it will allow easy access to the DELPHI data for the whole HEP community, most likely limited to the detector independent layer.

The fact that the I/O is separated from the analysis package and the layering of the information in structures allows access independent on the choice of the storage format. In particular one could imagine, with some effort, to use IDEA to access the DST of any $e^+e^-$ collider experiment. Some of the solutions adopted at implementation level regarding the treatment of simulated vs. reconstructed information are original and could be exported easily to an LHC environment.

The code has been developed having specific use-cases in mind. Presently the package can be used for analysis not too demanding on the particle identification level, QCD like, and work is ongoing to allow spectroscopy and searches analysis by the end of the year.

Being aware of the advantages of modern storage solutions which are based on object persistency, we would still prefer to maintain our data in ZEBRA format [4] in the case of the experiment data. It is assumed that IT division will care about the storage medium and about the definition of the storage format. If it will not be possible to maintain ZEBRA we need to know this before the end of this year, else we might lack the expertise to convert our DSTs to any different format.

# References


[1] F. Cossutti et al., *Improvements to SKELANA for Version 2.0*, DELPHI Note 99-175 PROG 239.

[2] G. Schneider and J. P. Winters, *Applying Use-cases*, Addison Wesly, 1998.

[3] N. Smirnov, F. Carena and T. Spassoff, *Object model of Delphi Data*, DELPHI Note 98-157 PROG 235.

[4] *The ZEBRA System*, CERN Program Library Long Writeups Q100/Q101.

[5] Objectivity Inc., `http://www.objectivity.com`

[6] T. Spassoff, *DELPHI extended short DST content*, DELPHI Note 97-147 PROG 222.

[7] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[8] P. Billoir and S. Quinn, *Fast vertex fitting with a local parametrization of tracks*, Nucl. Instr. Meth. Phys. Res., **A** 311 (1992) 139-150.

[9] N. Smirnov, F. Carena, and T. Spassoff, *Data archiving in the DELPHI experiment*, CHEP98, Chicago, 1998.

[10] T. Camporesi, P. Charpentier, M. Elsing, D. Liko, N. Neufeld, N. Smirnov, and D. Wicke, *Object oriented data analysis in the DELPHI experiment*, CHEP2000, Padova, 2000.