EPS-HEP99
Abstract # 15-707
Parallel sessions: 16
Plenary sessions: 15
ALEPH 99-056
CONF 99-031
June 30, 1999

P R E L I M I N A R Y

Object oriented data analysis in ALEPH

The ALEPH Collaboration and D. Düllmann[a]
[a] RD45 Collaboration, CERN, Geneva, Switzerland

Contact person : G. Dissertori, Guenther.Dissertori@cern.ch

Abstract

The project ALPHA++ of the ALEPH collaboration is presented. The ALEPH data have been converted from Fortran data structures (BOS banks) into C++ objects and stored in an object database (Objectivity/DB), using tools provided by the RD45 collaboration and the LHC++ software project at CERN. A description of the database setup and of a preliminary version of an object oriented analysis program (C++) is given.

Contribution by ALEPH to the 1999 summer conferences

1 Introduction

Object oriented (OO) programming is becoming the new paradigm for software development in high energy physics, in particular in view of the upcoming, very demanding LHC project and its experiments. In order to provide an analysis platform and OO application software, the LHC++ project has been set up at CERN [1]. Furthermore, commercial products for the storage of persistent objects in so called object databases are being evaluated by the RD45 collaboration [2]. The current candidate for a large scale application at CERN is Objectivity/DB [3].

In ALEPH a project called ALPHA++ [4] has been set up, with the following goals :

- convert the ALEPH data from a Fortran (BOS [5]) bank style into persistent objects and write them to an object database (Objectivity/DB)
- rewrite a mini-version of the ALEPH analysis package ALPHA [6] in an object oriented computing language (C++), based on the object database.
- compare the standard and OO performance with regard to the efficient access of the data and their manipulation.
- test the software engineered by the RD45 and LHC++ projects.
- provide input and experience for a possible archiving of ALEPH data.
- give an opportunity to learn OO analysis, design and programming.

In this report we present the structure of the ALEPH object database and a preliminary version of the analysis program.

2 The ALEPH data structure and its conversion to objects

ALEPH uses the Fortran based BOS system for the memory management. The event data are kept in memory in a large array which is globally accessible as a common block. The data are organized in so called *banks*, and the data definition language (DDL) for these banks is provided by ADAMO [7]. The ADAMO package offers a conversion to C header files, which facilitates the automatic conversion of the ALEPH DDL to C++ classes, or to be more precise, to the ddl-files needed for the setup of an Objectivity/DB database. It has been decided to perform a one-to-one conversion of all the relevant banks stored on an ALEPH DST (data summary tape), which can be read by the analysis program ALPHA. A simple C++ tool has been developed which allows for the automatic conversion of the ADAMO DDL of all 173 relevant banks into C++ classes. An example of the correspondence between the ADAMO description of the track bank FRFT and its C++ implementation is given in Fig. 1.

The general DDL structure as implemented in the Objectivity database is outlined in Fig. 2. A class *AlephBank* serves as a base class for all the ALEPH banks. Persistency is obtained by inheritance from the Objectivity class *ooObj*. The class *AlephBank* stores the name of each bank, and defines the interface for the reading from and writing to memory of the bank contents. Each

ADAMO DDL

```
FRFT
    :   'Global Geometrical track FiT
        NR=0.(JUL)\
        Number of words/track\
        Number of tracks'
   STATIC
    = (InverseRadi   = REAL [*,*],
       TanLambda     = REAL  [*,*],
       Phi0          = REAL  [0.,6.3],
       D0            = REAL  [-180.,180.],
       Z0            = REAL  [-220.,220.],
       Alpha         = REAL  [-3.15,3.15],
       EcovarM(21)   = REAL  [*,*],
       Chis2         = REAL [0.,*],
       numDegFree    = INTE [0,63],
       nopt          = INTE [0,149]
);
```

C++ CLASS

```
class FRFT {
public:
// default constructor
  FRFT() {}

  float  InverseRadi;
  float  TanLambda;
  float  Phi0;
  float  D0;
  float  Z0;
  float  Alpha;
  float  EcovarM[21];
  float  Chis2;
  int  numDegFree;
  int  nopt;
};
```

Figure 1: Correspondence between the ADAMO description of the track bank FRFT and its C++ implementation.

ALEPH bank is implemented as a class *NAME_Bank*, where *NAME* is the BOS name of the bank, such as FRFT. This class *NAME_Bank* contains a variable length vector *NAME_Table*, the elements of which are objects of the class *NAME* obtained from the conversion of the ADAMO DDL into a C++ class. For example, the elements of the vector *FRFT_Table* are the FRFT objects (tracks) of a given event. *NAME_Bank* also has the specific implementation of the read and write methods for the memory access.

3 The ALEPH object database

The database structure of the ALEPH object database is reproduced in Fig. 3. At the basis there is a federated database called ALEPHDB, which contains several databases. These databases either contain real data from different data taking periods with or without pre-classification of events, or Monte Carlo events from fully simulated hadronic Z decays.

Every database has a container holding *Run* objects. These objects store the run number and have bi-directional links to all the event objects of the corresponding run and to banks which store global run condition data. Furthermore, for every run there is a container with the event objects of this run and an additional container with all the bank objects per run. The event objects store the event number and the classification bit of the event, and have bi-directional links to all the banks of the event.

At the moment about eight Gbyte of real and Monte Carlo data are stored in the federated database. In the following some examples are given. A database which contains 5 runs, 19773 events without preselection and a total of 374686 banks, has a size of 234.3 Mbytes. The operation of the Objectivity tool `ootidy` reduces this by 5% to 221.2 Mbytes. The size of the corresponding EPIO file where the BOS banks are stored is 186.2 Mbytes. The overhead arises mainly from the relationships which are stored in the object database. A high page density of 99.8% is achieved.

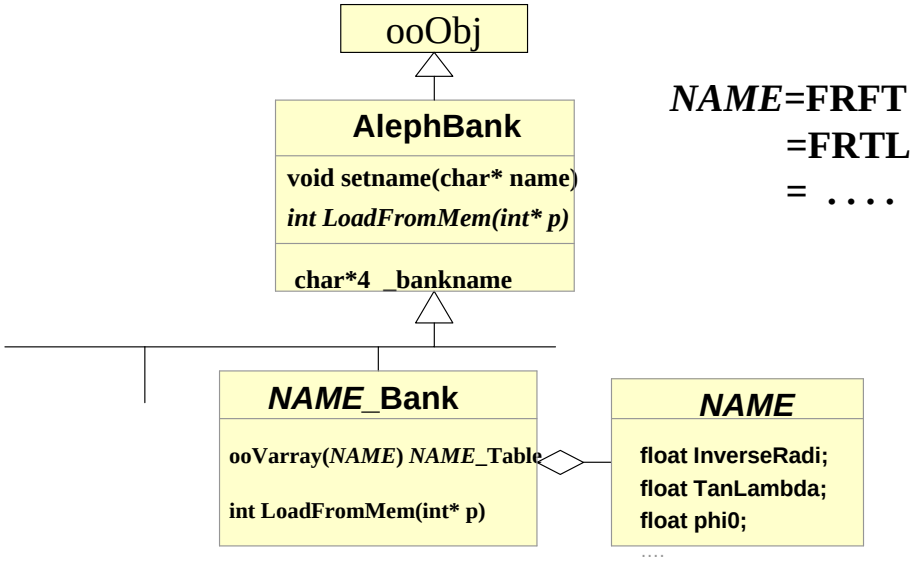

Figure 2: The DDL structure of the ALEPH object database

Another database of 19 runs, 5000 events classified as hadronic events and 1204362 banks, has a size of 763.4 Mbytes, whereas a database with 5000 simulated hadronic events is of similar size, containing 883047 banks.

For the moment two simple C++ programs exist for the reading and writing of the databases. In the reading case, a loop over all run, event and bank objects is performed, using the iterators over the bi-directional links, and for every bank in the loop its method is called which copies the bank contents into memory in order to restore the BOS common. This allows for the application of the standard Fortran based analysis program ALPHA as well as for the event display program to be used. The banks stored in the memory are also the starting point for an object oriented analysis program which is described in the next section.

4 The Analysis Program

The ALPHA++ analysis program is based on the structure of ALPHA. Two basic objects are defined: the *Tracks* and the *Vertices*.

A *Track* can be a charged track reconstructed in the TPC, a photon identified in the ECAL or a general Energy Flow Object. The common attributes of a *Track* are: the total momentum with its components, the energy, the mass and the charge. A *Track* can also have more specific attributes depending if it is a charged track, a photon or an Energy Flow Object.

The previously described structure has been implemented in ALPHA++ with an abstract class *AlObject* (Figure 4), with the common attributes of a *Track* defined as purely virtual member functions. The concrete transient classes *Altrack* (charged tracks), *AlEflw* (Energy Flow) and *AlGamp* (photons) are defined by inheritance from *AlObject* (Figure 5).

The same principles apply also to the definition of the *Vertices* (Figure 4). There is an abstract class *AlVertex* with the basic attributes, and two concrete transient classes *AlMainVertex*, holding

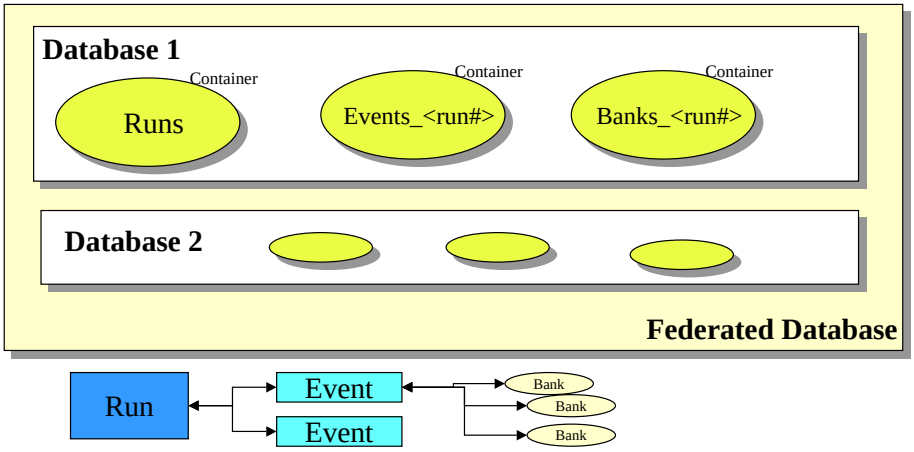

Figure 3: The database structure of the ALEPH object database.

the position of the interaction point, and *AlGenVertex* for the secondary vertices found by the reconstruction program (Figure 6).

The implementation of the previously mentioned transient classes in ALPHA++ makes use of the ALPHA internal banks QVEC (containing the *Tracks*) and QVRT (containing the *Vertices*). This is done in the following steps:

- For each event the relevant persistent classes from the Objectivity database are read and the corresponding BOS banks in the Fortran BOS common are filled;
- the Fortran subroutines which starting from the BOS banks construct the internal QVEC and QVRT data structures are called from C++;
- the concrete C++ classes (such as AlTrack, AlMainVertex etc.) are instantiated using the data contained in QVEC and QVRT;
- for each event the transient class *AlphaBanks* containing all the instances of the previous classes and giving access to them through member functions is instantiated.

A complete class diagram of the analysis program is shown in Fig. 7. The choice to call Fortran from C++ in the analysis program is due to the fact that there are many algorithms in ALPHA which, on a short time scale, are not possible to develop in C++. Therefore our strategy is to have a "Fortran wrapped" analysis program already working and to use it as a basis to develop new C++ code and algorithms.

5 Conclusions

A project called ALPHA++ has been set up within the ALEPH experiment in order to test new object oriented approaches to data storage and analysis. The data definition language, the structure of the object database as well as a preliminary version of an object oriented analysis program have been presented. Future work will concentrate on the further development of this analysis program as well as on detailed performance studies.

6 Acknowledgements

We would like to thank the members of the RD45 collaboration and of the LHC++ project for their help with respect to conceptual as well as technical problems.

References

[1] http://wwwinfo.cern.ch/asd/lhc++/index.html

[2] http://wwwinfo.cern.ch/asd/rd45/index.html

[3] http://www.objectivity.com/

[4] http://alephwww.cern.ch/~disserto/objty/Welcome.html

[5] http://www.cern.ch/Light/examples/aleph/bos/bos.html

[6] http://alephwww.cern.ch/ALEPHGENERAL/phy/doc/alguide/alguide.html

[7] http://www1.cern.ch:80/Adamo/refmanual/Document.html

```
class AlObject {
 public:
  ~AlObject();
  virtual float QP() = 0;
  virtual float QX() = 0;
  virtual float QY() = 0;
  virtual float QZ() = 0;
  virtual float QE() = 0;
  virtual float QM() = 0;
  virtual float QCH() = 0;
};
```

```
class AlVertex {
 public:
 ~AlVertex();
  virtual float VXposition() = 0;
  virtual float VYposition() = 0;
  virtual float VZposition() = 0;
  virtual int Vertex_number() = 0;
  virtual int Vertex_type() = 0;
  virtual float ChiSquareFit() = 0;
  virtual float* CovMatrix() = 0; //
     pointer to covariance matrix
};
```

Figure 4: The AlObject and AlVertex abstract classes interface

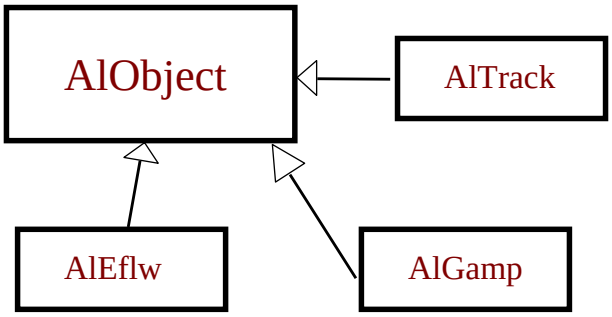

Figure 5: The AlObject inheritance structure

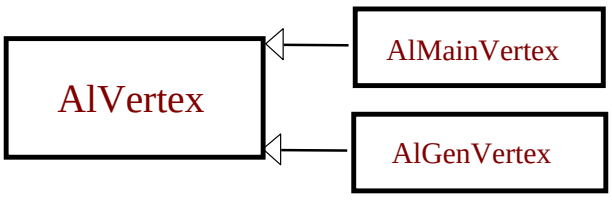

Figure 6: The AlVertex inheritance structure

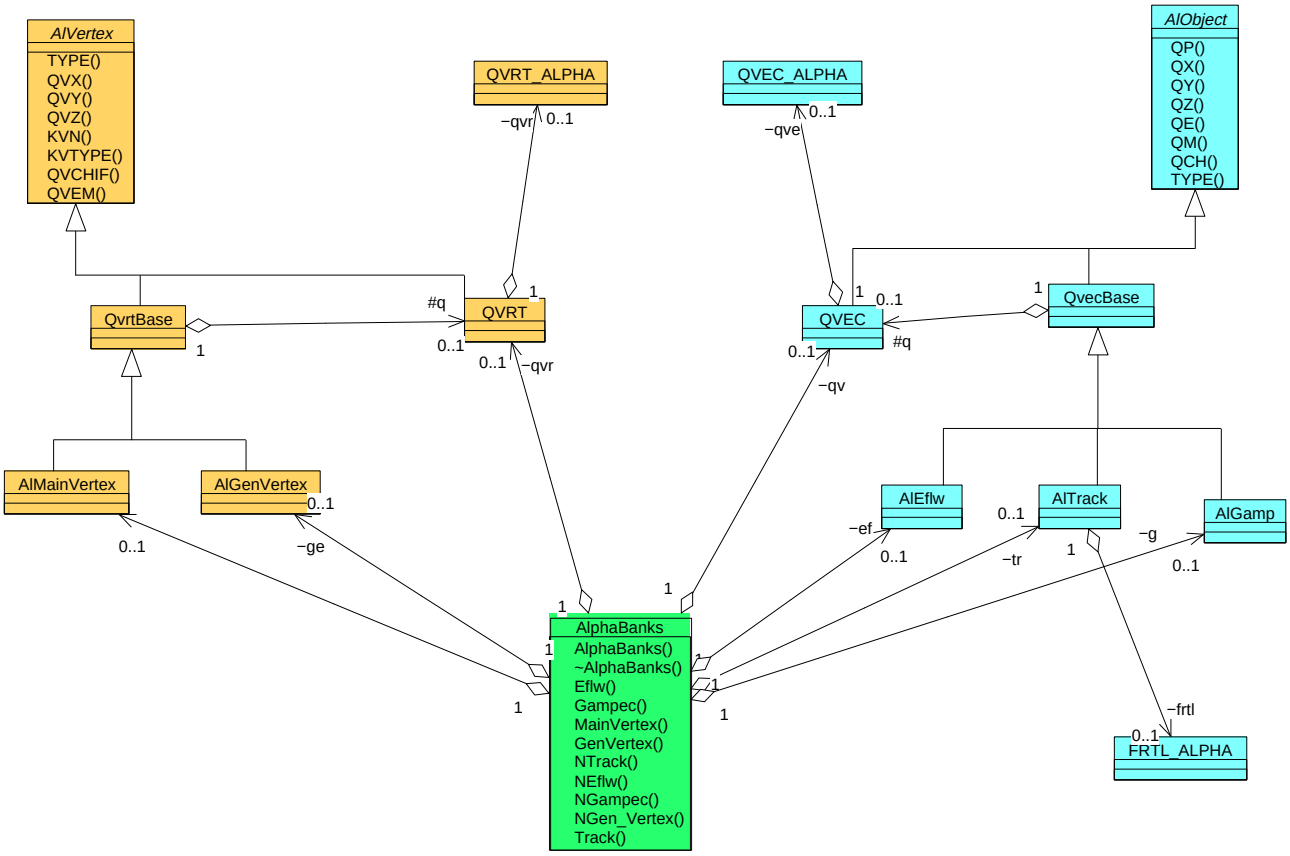

Figure 7: ALPHA++ analysis program class diagram (reverse engineered). The *vertex* related classes are on the left side, the *track* related classes are on the right side. The class *AlphaBanks* acts as a container for the previous classes.