Object Oriented reconstruction for the Instrumented Flux Return of BABAR

Luca Lista for the BABAR Computing Group

INFN Sezione di Napoli
Complesso Universitario di Monte Sant'Angelo, edificio G, I-80126, Napoli, Italy

Abstract

The application of Object Oriented design patterns to the reconstruction software of the Instrumented Flux Return detector of BABAR experiment is presented. The use of abstract interfaces improved the developement of reconstruction code and permitted to flexibly apply modification to reconstruction strategies, and eventually to reduce the maintainance load. The experience made during the last years of developements is presented.

1 The Instrumented Flux Return

The Instrumented Flux Return (IFR)[1] system identifies muons and neutral hadrons in the BABAR experiment[2]. The active elements of the detector are a number of layers (19 in the barrel, 18 in the endcaps) of Resistive Plate Chambers (RPC) inserted in the gaps present in the segmented iron yoke of BABAR magnet. Two layers of cylindrical RPC inserted inside the coil complete the system. Each layer is equipped with two strip planes which produce a digital pattern. The reconstruction creates a charged cluster, which is candidate for a muon or charged hadron signal, from the strips associated to a charged track. The strips unassociated to charged tracks are clustered to form candidates for neutral hadron signal.

2 Recontruction Objects

Reconstruction information is modeled as objects in BABAR reconstruction software. The most basic object produced by the IFR detector is the strip. Adjacent strips in the same readout view are clustered into one-dimension (1D) cluster objects, also referred to as hits. Two different clustering algorithms for charged and neutral particles produce cluster objects. The IFR is segmented in sectors. The barrel is divided in 6 sectors along ϕ, each endcap half door is divided vertically into 3 sectors, while the cylinder is divided in 4 sectors. Due to this segmentation, we define basic three-dimension (3D) cluster objects that model a cluster with information contained in a single sector; a composite pattern[3] is applied to define clusters which contain information in more than one sector. A 3D cluster reconstructed in a sector with planar geometry (barrel or endcap) is composed of two two-dimension (2D) cluster objects reconstructed in each of the orthogonal readout views, spanning many layers of the sector. A cluster reconstructed in the cylindrical RPC is composed of 2 to 4 hits in the four stereo readout views. The stereo readout permits a reduction of the "ghost" intersections when more than one track is present in one sector. All 3D cluster types inherit from the same base class. This shields the clients from implementation details. This kind of class organization permits to compute most of the interesting cluster quantities in the coordinate system local to the sector or readout view, then combine the results in the composite object using a recursive computation.

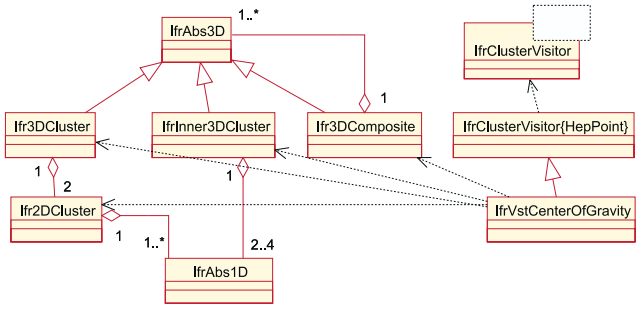

Figure 1: Class diagram of the IFR clusters and visitor classes.

The client can request the needed information, like the cluster centroid, number of strips fired, number of layers hit, cluster covariance matrix, etc., using a visitor pattern[3] (fig. 1). New functionality of the cluster classes can be introduced without changing any of the cluster class interface with the addition of a new concrete visitor subclass[4].

An example of the code that computes some cluster quantities is the following:

```
IfrAbs3D * cluster; // get a cluster from somewhere
int      n = cluster->accept( ifrVstInt  ("numberOfStrips" ) );
HepPoint p = cluster->accept( ifrVstPoint("centerOfGravity") );
```

The functions `ifrVstInt`, `ifrVstPoints` fetch from an hash map a specific visitor based on a string key. In this way, the client code needs only to depend on the abstract cluster and abstract visitor base classes. The classes that define clusters and visitors are separated in different packages. Base classes (`IfrAbs3D` and `IfrClusterVisitor`) are contained in the package `IfrData`, while all concrete cluster types are in package `IfrDataImp`, and all concrete visitors are in the package `IfrVisitors`. Clients of the IFR reconstruction only need to depend on the most stable package `IfrData`, with no dependency on the package `IfrVisitors` that is most frequently changed.

3 Access to geometry information

An IFR detector model is needed in order to access the detail of the geometry information, like strip pitch, layer orientation and position in the space, corrected by alignment calibration. The IFR detector model is organized in a composite structure: the IFR detector is composed of different sectors, each made of different layers, each with two readout views, each with different Front End Cards. A composite pattern has been applied to encapsulate the basic abstract functionality, like coordinates transformations from local to global and viceversa, and the navigation tools along the detector tree. Specific smart iterators permit to iterate on all the subcomponents of a given structure at an arbitrary depth in the detector tree. The classes have been organized in two packages: `IfrGeomBase`, which contains the base classes of the composite pattern (`IfrDetComponent`, `IfrDetComposite`, `IfrDetLeaf`) and the base iterator, while the package `IfrGeom` contains all the other subclasses. Most of the clients need to depend on `IfrGeomBase` only; still some limited dependencies on the `IfrGeom` subclasses are needed because some functionality specific to the concrete component cannot be abstracted in the base class (like the strip pitch information, which belongs to the view).

4 Cluster Reconstruction

Two different algorithms are applied to reconstruct charged and neutral clusters in the IFR. The clustering algorithms are implemented as modules of the BABAR Framework[5]. First, all charged tracks are extrapolated in the non uniform magnetic field map. The intersection of the track extrapolation with the RPC layers are computed, and all the hits distant from the expected intersection less than a given cut are associated to the track. The associated hits are passed to a specific object (`IfrClusterizer`) that has the task, independent on the clustering algorithm, of organizing the 1D clusters by sector and by view, and arrange them into a specific type of cluster (single planar or inner, or composite), according to the hit pattern. All hits remaining unassociated to charged tracks are then processed for neutral clustering. The neutral clustering algorithm first creates 2D cluster objects associating hits of the same readout view which are spacially closer than a maximum distance cut in that projection (fig. 2), then the 2D clusters of the same sector are combined in a 3D cluster object if the hit layers match the two projections within some tolerance. A small fraction of ambiguous associations may survive; the ambiguity can be suppressed if further information is provided (like the expected K_L direction), and the update of the ambiguity status of all the clusters is handled automatically when a cluster is forced externally to become unambiguous.

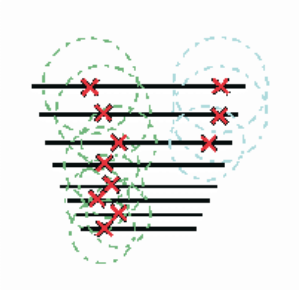

Figure 2: Adjacency criterion for 2D clustering.

The inner RPC has a specific reconstruction algorithm that tries all possible combinations in the same sector, trying first to combine 4 views, keping the combination with the best χ^2, then trying the 3 views combinations on the remaining hits, then 2 views combinations.

It is also possible to create composite neutral objects applying an adjacency cirterion on the neutral components reconstructed in different sectors.

Framework modules contain the steering code for 2D (and 3D) clustering, including the details of iterating over sectors, views, etc., and use abstract cluster finder objects which abstract the basic functionality of performing clustering on all hits of a given view (or sector). Different concrete subtypes can implement specific algorithms, as in a strategy pattern[3]. The different algorithms can be selected at run time.

In an early stage of the reconstruction developement, before the track extrapolation tools where available, the neutral clustering algorithm was applied for charged cluster finding as well. The clusters found in this way where matched to the charged tracks and composite objects where created combining all clusters associated to the same charged track. Moving to the current charged clustering algorithm didn't require any change to the available cluster classes and the algorithm could be completed in a time shorter than what was initially scheduled. The cluster representations and the way visitors extract information from the clusters in fact do not depend on the algorithm used to build the cluster, and this permits easy improvements of the reconstruction algorithms, and preserves external client code unchanged because no change is induced in the class interfaces.

5 Particle identification

Particle identification is made using discriminating variables extracted from the cluster. The different distribution of variables like number of interaction lengths, last layer reached, and average strip multiplicity permit the rejection of pion background for muon identification.

Such quantities are computed on different cluster types using visitor classes. In some cases, specific visitor classes need extra information for their computation; for instance, the visitor that computes the χ^2 of the match of the cluster with a charged track needs to receive the track extrapolation as input to access the intersections with the RPC layers.

A summary object which contains the most relevant quantities is delivered to the user code. This `IfrPidInfo` inherits from a common BABAR base class, and its functionality is described in a separate presentation[6].

6 Software developement and design evolution

Several migrations have occourred in the IFR software before the currently used version. Sometimes we saw problems occourring repeatedly in the same code areas as new features where introduced. In those cases, the application of a more flexible design has always carried benefit towards more effective and safe developement, and isolating problems. This was the case of the introduction of the visitor rather than the simple usage of virtual functions in the base class.

The usage of a polymorphic design improved the developement process. For instance, during the developement of the code for the computation of the number of interaction lengths traversed in the iron using a fit to the cluster hits, we introducted abstract classes to model a curve approximation of the cluster reconstructed in 2D and 3D. In this was, the actual computation using the detector model was tested using a simple straight line fit as concrete implementation, and was developed in parallel to the polynomial fit implementation, coded as a separate subclass of the curve approximation. The integration of the two components was immediately successfully.

References

1 The Muon and K_L Detector for the BABAR Experiment: Physics Requirements, Final Design and Start of Construction, Nucl. Physics B (Proc. Suppl.) 61B (1998) 244-249

2 BABAR Technical Design Report, BABAR Collaboration, SLAC Report SLAC-R-95-4578, March 1995

3 Design Patterns, E. Gamma, R. Helm, R. Johnson, J. Vlissides, Addison Wesley 63361, 1995

4 L. Lista, proceedings of CHEP '98: "Evolution of cluster design in the Instrumented Flux Return (IFR) of BaBar", August 31 - September 4, Chicago, USA.

5 E.D.Frank, R.G.Jacobsen, E.Sexton-Kennedy, proceedings of CHEP '97: "Architecture of the BaBar reconstruction system", 7-11 April 1997, Berlin.

6 G. De Nardo, L. Lista, proceeding of CHEP 2000: "Object Oriented design of Particle Identification Software for Instrumented Flux Return subsystem of the Babar detector", 7-11 February 2000 Padova.