OORICH Design Report

LHCb Technical Note

Issue: Draft
Revision: 1

Reference: LHCB XXX 2000
Created: 10 March 2000
Last modified: 29 March 2000

Prepared By: Dietrich Liko
Niko Neufeld

Abstract

The design of the object oriented RICH reconstruction software for LHCb is reviewed in detail. A first implementation has already been provided and it has been found to fulfil the requested requirements. The design is now already in the second stage of an iterative design process. To develop a useful tool for the future the program has to be developed both in terms of “physics features” and in conceptual design. The analysis is starting with a brief review of the user requirements. Following the UML development process entities are identified. These entities are then analysed in terms of individual function and interaction between them. On that basis a design is developed that allows the evaluation of the use cases. Finally a component-based architecture is proposed. This will provide the means to create a flexible software system that has the high performance necessary for the reconstruction of the RICH data.

Document Status Sheet

Table 1: Document Status Sheet

1. Document Title: OORICH Design Report			
2. Document Reference Number: LHCB XXX 2000			
3. Issue	4. Revision	5. Date	6. Reason for change
Draft	1	23 March 2000	First version

Table of Contents

List of Figures

List of Tables

1. Introduction

This document outlines the status of the object oriented offline reconstruction of the LHCb RICH[1]. The project started originally as an attempt to apply modern software design principles to an offline reconstruction program. While the algorithm was based on the FORTRAN algorithm [2] used to provide results presented in the Technical Proposal, the structure was newly developed. On the same level it was also important to write a program that was competitive with the already existing algorithm in terms of physics performance and resource consumption. This is an area where object-oriented solutions are well known to have difficulties. The program has been subsequently adapted to the GAUDI environment.

In this document we review the current design and present now how we would envisage the organization of the software for the next release.

1.1. General Considerations

Figure 1 shows the principle of execution of algorithms in GAUDI. The algorithms are chained and one algorithm after the other is modifying in turn the data inside the data store. Consequently possible dependencies are only via the data in the store. GAUDI itself has few requirements on the implementation of the algorithm. As a matter of fact it is envisaged that for some time also FORTRAN algorithms will be used.

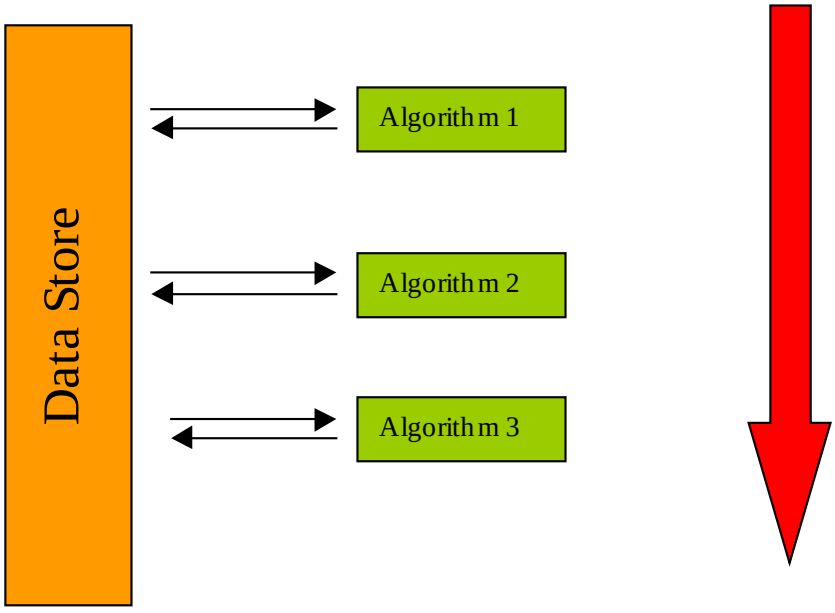

Figure 1: GAUDI Algorithms interacting with the data store

Based on this model we want to develop a modular system for RICH reconstruction. Our aim is to profit from object-oriented techniques in the design and implementation of these modules. For that purpose a careful study of the requirements is necessary. Based on this requirements the development process according the UML technique is then followed.

A number of additional considerations have to be taken into account for such a system

- Functionality
- Consistency
- Maintainability

We think that such requirements are most easily implemented by a system of components with well identified and isolated functionallity. These components will be assembled to provide the necessary functionality for the algorithms. This picture has to be compared with a set of independent algorithms.

Functionality implies that the components need to have the necessary complexity to perform the task. The reuse of components in different algorithms is enforcing consistency. Components can also easily be exchanged and therefore maintainability is improved.

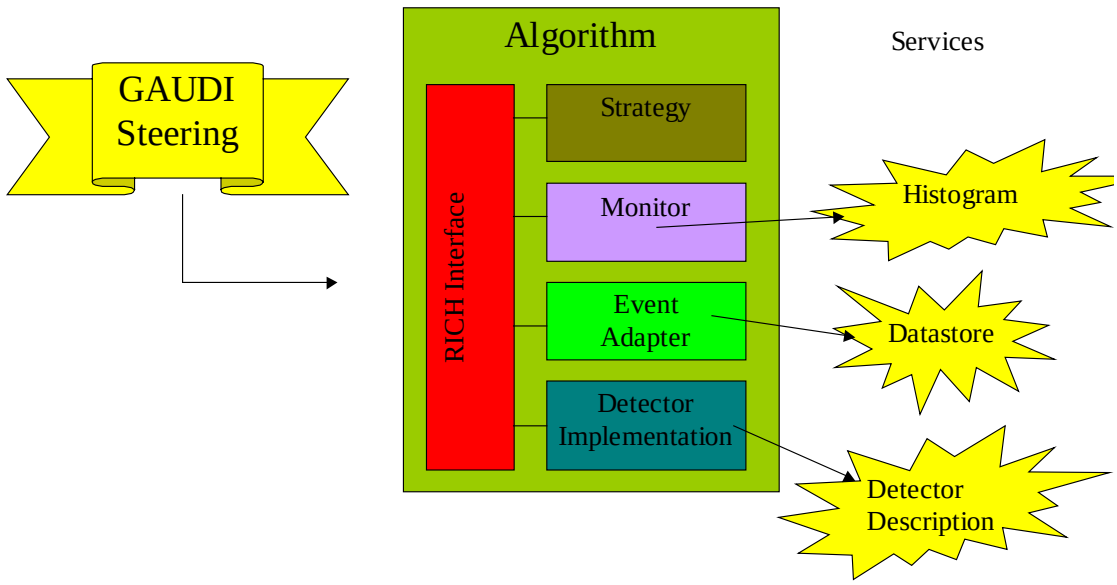

Figure 2: Components of RICH the reconstruction software

An important additional ingredient of such a system is a common interface that is separating the components. It can also provide functionality common to all algorithms. As discussed in more detail later we can identify following components

- RICH Interface
- Algorithm Strategy
- Algorithm Monitor
- Detector Implementation

- Event Adapter

The architecture is shown in Figure 2. Here also the interaction of the various parts with the GAUDI framework is shown.

The common "RICH interface" is a GAUDI Top Algorithm. According to settings in the Parameter File it is creating instances of the other components. The GAUDI framework provides flexible instantiation of these components. This requires these components to be independent sub algorithms. After instantiation of the component a down cast unveils the application specific interface. This is discussed in more detail in Chapter 5.

1.2. Current Status of the Implementation

A standalone version of the Rich Reconstruction program has been presented last November[4]. For this version a detailed comparison was performed in terms of physics performance and resource consumption.

The comparison was performed between the standalone C++ program and the standalone FORTRAN program that has been used for the numbers quoted in the Technical Proposal. The evaluation was performed using 100 simulation events of $B \rightarrow \pi^+\pi^-$ decays and background. Input for both programs was the same ASCII file containing information on particle entry points that were based on the GEANT 3 simulation. In both cases photon simulation was performed and the particle hypothesis was reconstructed using the same algorithm. This provides a test on compatibility with the old implementation.

Table 2: Reconstruction Performance of the standalone program

Rec	True						P
	e	μ	π	K	p	X	
e	8848	7	426	3		51	0.95
μ	20	230	1163	3		57	0.16
π	8	13	10891	9		29	0.99
K	2	1	39	1083		11	0.97
p	1		4	1	427	1	0.98
X	154	2	67	12		9700	0.98
ε	0.98	0.91	0.87	0.97	1.00	0.99	

In Table 2: Reconstruction Performance of the standalone program the reconstruction performance is given for those 100 events. The performance has found to be similar to the FORTRAN program and the values quoted in the Technical Proposal [1]. A typical effect, the low purity of the muon tag, is also reproduced.

The CPU time consumed by the program has been found similar to the FORTRAN program (see Figure 3, the quoted seconds are CPU seconds). This was considered good news, because the overhead introduced by C++ language features has therefore to be not too large.

An attempt has been made to measure the improvement obtained by using a commercial C++ compiler[5]. The expected gain in performance has been observed.

Afterwards the program has been adapted to the GAUDI environment. A first release (v1r1) is available via CVS[1]. Following modifications have been performed:

- GAUDI algorithms and steering via standard Parameters
- Histogramming and logging mechanism
- Detector geometry is based an a ASCII file
- Strategy and Monitor Objects based on GAUDI sub-algorithms
- Adaptation for Windows NT and as a Windows DLL

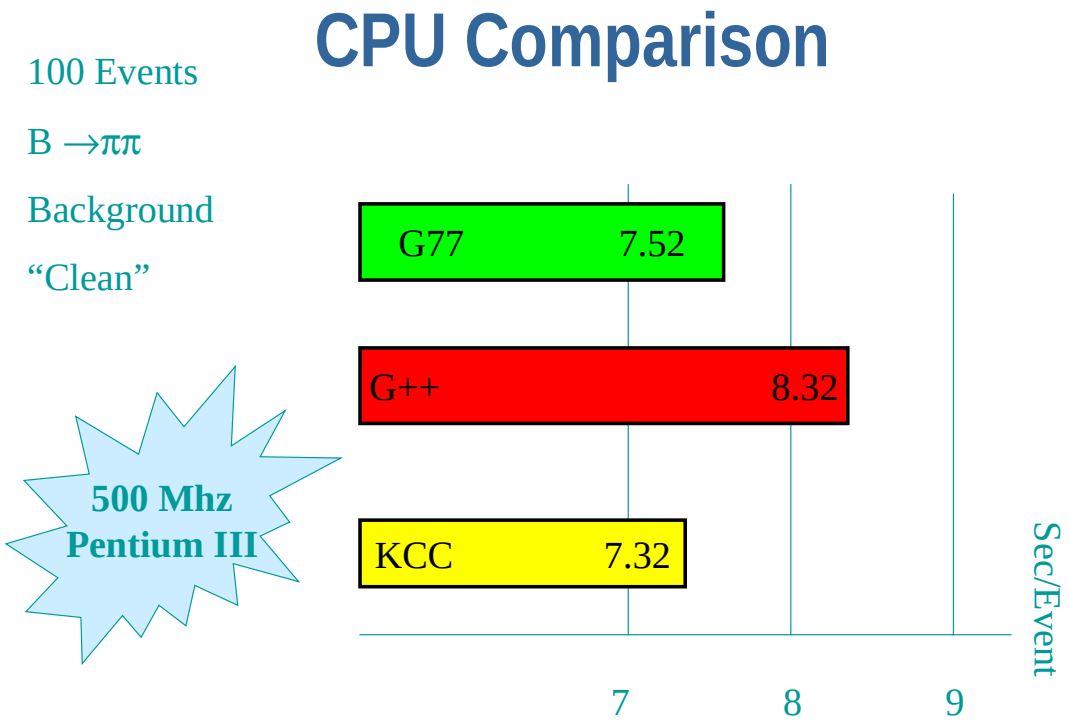

Figure 3: Comparison of CPU consumption for the FORTRAN and the C++ program

[1] To obtain a working version use the LHCb getpack command: `getpack OORICH v1r1`

1.3. Future development

Consolidation of the object-oriented architecture: This is following the spirit of iterative development cycle of the UML development process. After the first implementation has been developed the cycle has to be iterated in all phases. After a working version of the program has been obtained it is important to understand possible flaws in the design at the first place. This allows improving the abstractions provided by the entities or the classes. As a matter of fact this document is part of the process, as it describe the current status of the program in the light of the development for the next release.

Better Adaptation for the GAUDI environment: Of particular importance is the implementation of the object-oriented detector description. This needs not only development within the program, but also some planning of the information flow in the future. This concerns in particular the conversion of information stored in the so-called CDF files into the XML format.

Implementation of new features: Obviously the program has to be developed to be a useful tool for future analysis. This requires the implementation of similar features as are already implemented in the current version of the Fortran program. A number of points come to mind:

- Update of the geometry
- Use of reconstructed particles as seed for the photon reconstruction
- Realistic Photon Detector Implementation
- Adaptation of the log likelihood function to background terms

In the following chapters the requirements are reviewed (Chapter 2). Special emphasis is given to a description of important Use Cases. Based on the analysis of the requirements a list of entities is derived (Chapter 3). They are then incorporated into a design of interacting entities (Chapter 4). This design is finally realized by this architecture (Chapter 5). Some special Use Cases are studied using Interaction and Sequence Diagrams (Chapter **Error! Reference source not found.**). Finally conclusions are drawn and an outlook is given (Chapter 6).

2. Requirements

A fundamental part of the object-oriented program development process is the capture of the user requirements. It has been noticed that this is a non-trivial task in the area of detector reconstruction programs. This is partially due to the nature of the problem and partially due to the way physicists are used to describe it. Phrasing the problem in terms of questions or commands seems not to be right, as we seem to know anyway, how the program should solve the problem.

Nevertheless way have applied a "Use case" analysis to a part of the problem specification. We have found that this way of describing the problem is very useful, because it leads to a new way of thinking about the problem. It allows describing the algorithm by a number of interacting "physics" entities. They are the basis of the construction of a truly object-oriented system that can be used to implement the algorithm

Other parts of the problem specification, as detector description or physics formulas, are better described in the traditional way and some aspects are briefly discussed in the corresponding chapter.

2.1. Use Case Analysis

We have found a way to use the "Use Case" description for part of the problem: We replace the main steering algorithm by an imaginary "Physicist". This physicist takes the role of the "Actor", who wants to solve the problem, e.g. wants to simulate or reconstruct an event. The physicist has to ask questions to imaginary "physics" entities to provide the answers necessary to derive the solution. By that requirements for this entities are derived, which are the basis for the implementation.

In praxis this leads to an extension of the concepts of "well known" entities, like particles or detector hits. Within the domain of the relevant algorithms additional questions can be asked or additional actions can be performed. As an example we can consider a "Track" entity: Firstly one expects a track to know about its momentum, of detector participating in the reconstruction etc. In addition we expect the "Track" entity in the RICH domain to provide also RICH specific information, as the number of emitted photons inside a particular radiator. The use cases are used to construct a complete system of such questions.

2.1.1. Global Likelihood evaluation

The solution for the association particle types to all particles of an event is derived in [2] . It is given as the solution of a global likelihood function.

$$\ln L = -\sum_{j=1}^{N} \mu_j(h_j) + \sum_{i=1}^{M} n_i \ln \sum_{j=1}^{N} a_{ij}(h_j) - C$$

The solution is found by variation of a the hypothesis h_i for each track. The minimum of the function is searched.

We can identify two important terms: One term is question to each track, $\mu_i(h_i)$, for the expected number of photons to be observed in the photon detector. This is a complex question which involves the

calculation of the expected number of phhotons to be emitted by the track segement in the photon radiatior for a particular particle type hypothesis. The second term is a geometric efficiency terms, which needs calculation by Monte Carlo integration.

The second term is a question for each hit pixel: How large is the observed signal and what signal is expected for a particular global hypothesis on this particular pixel. This term needs calculation of all possible contribution to a pixel by all track under their particle hypothsis. This includes calculation of a hypthetical emission angle. Taken into account the resolution of the individual photon, this contribution can be evaluated.

A grafical repesentation of the some questions to the track entity is shown in Figure 4.

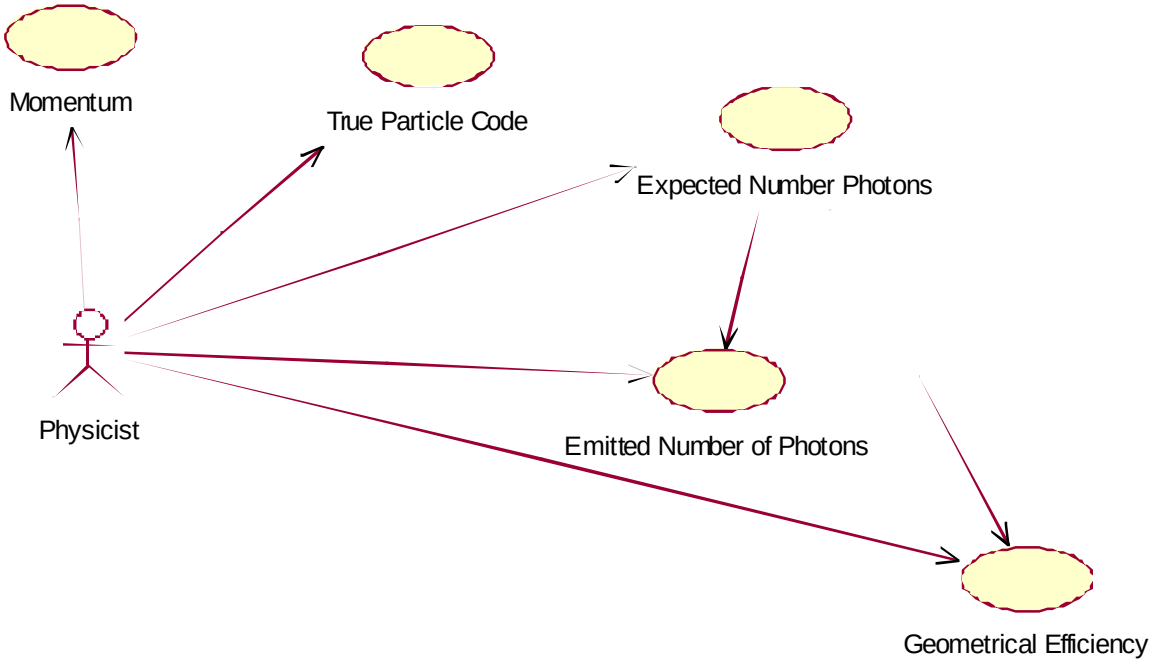

Figure 4. Use Case description of a track

2.1.2. How to simulate photons

For each particle in turn the average number of photons to be emitted is calculated. The number depends on the

- particle type,
- particle momentum,
- length of the track segment in the radiator,
- refractive index of the material including the dispersion,
- energy window given by the photon detector and a possible exit window of the radiator,

In addition it is of advantage to take into account the quantum efficiency of the photon detector. This introduces an energy dependence in the distribution of the photons. This can significantly reduce the CPU consumption during simulation, as a large fraction of the photons are finally not observed.

The number of emitted photons to be simulated is then given by a poisonian random number.

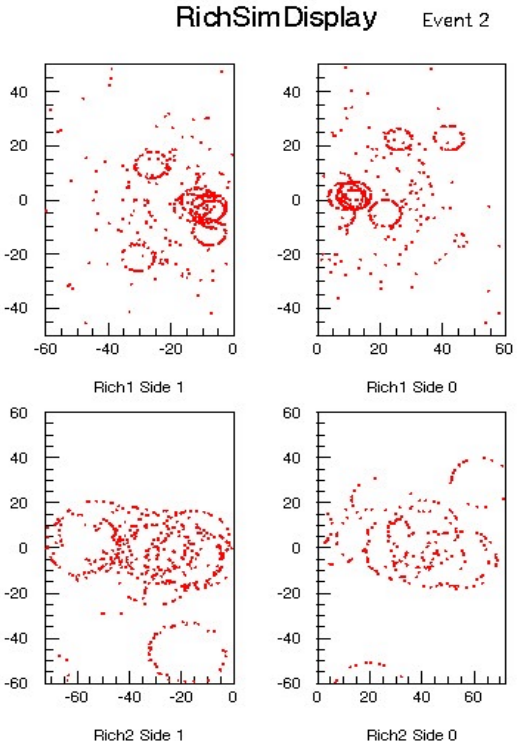

Figure 5: A display of a simulated event

The photons have then to be simulated in turn. This includes several random numbers for the physical properties of the emitted photon, including its

- energy,
- emission point along the track segment
- azimutal angle

The cherenkov angle itself is defined by the energy, the particle type, the momentum and the dispersion of the radiator material. In aerogel additional scattering has to be taken into account. Finally the photon has to be propagated inside the detector. Absorption at various edges is possible. The photon gets reflected on the mirror and is finally intersected with the photon detector plane. A final simulation of details of the photon detector is necessary.

In addition additional sources of background have to be considered as particles directly entering the photon detector or electronic noise.

The most important requests to the track segment entity are then

- How many photons to be emitted

- Simulate a photon

Both questions require a difficult interaction with other entities to derive the answers. The summary of all questions is presented in the next section.

2.2. Detector description

Not all information is well described by a use case analysis. Some information is better documented by straight forward listings of properties or formulas. The detector description is not given in all details, but is rather shown as an example.

The description of the detector device is naturally of fundamental importance. For this a simple description of the various parts and their physical properties seems the most adequate way to define the User Specifications. At this place no attempt is made top describe the various parts in detail. To give the reader an impression pictures of the devices from the Technical Proposal are included.

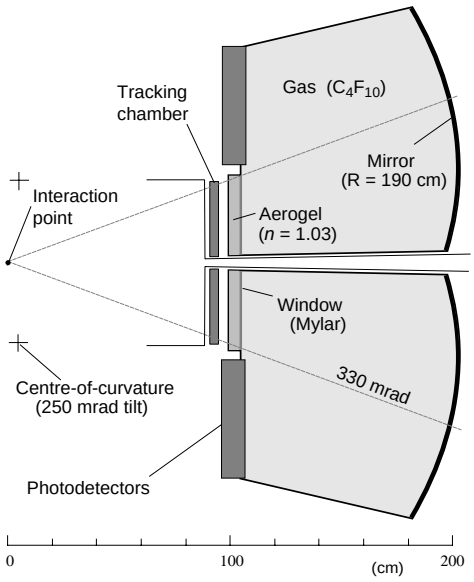

Figure 6: RICH 1 Picture from the Technical Proposal

The most important components of the RICH detectors can be classified in three groups

- Radiator System
- Mirror System
- Photon Detector

For RICH1 there are two radiator systems, an aerogel radiator and a gas radiator (see Figure 6). Both are similar in functionality, but naturally quite different in geometry. In addition there are additional physics processes in the aerogel that are important.

The mirrors form a simple system that are focusing the photons on two planar photon detector plans left and right of the beam tube.

For RICH2 there is only one radiator system, the gas radiator (see Figure 7). On the other hand there is a more complex mirror system with an additional flat mirror.

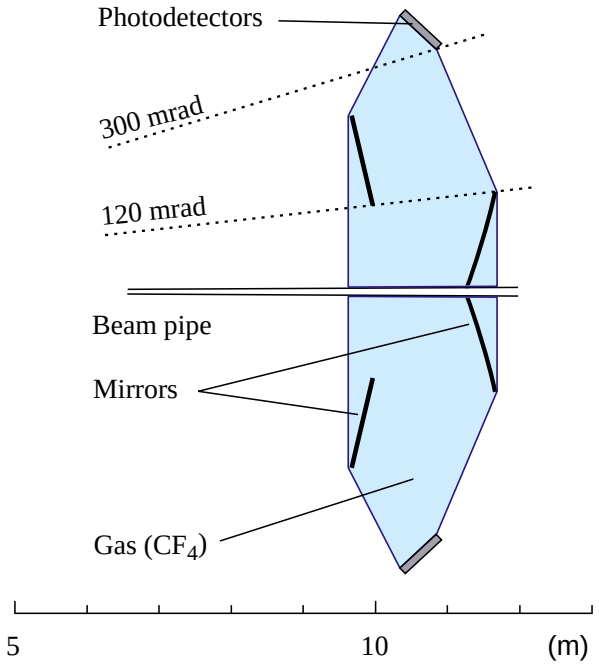

Figure 7: RICH2

A further important aspect is the photon detector. Here the current FORTRAN program is clearly more advanced. In the object oriented program a generic and simplified device is simulated. More detailed implementation will have to be developed in the future.

An important technical detail is that the geometrical specifications of the FORTRAN program are transmitted using CDF files. For the object oriented program XML files are foreseen. This will require the development of a specific translator program. Currently the C++ program uses a simple ASCII file as input for the detector description.

2.3. Physics Processes

A number of physics processes have to be considered during simulation and reconstruction. Further there are several algorithms given to perform various reconstruction tasks. They are part of the detailed

specifications. As an example only the Cherenkov effect is mentioned here. Other formulas will be documented in a future User Specification Document.

Cherenkov Effect

The particle is passing through a radiator with given refractive index n, with a velocity β exceeding the speed of light in the medium. The photons are emitted under a constant Cherenkov angle

$$\cos\vartheta_c = \frac{1}{n\beta}$$

The number of photons produced for a photon energy interval ΔE and path length l is given by

$$N = 370 \times \sin^2\vartheta_c \times \Delta E \times l = 370 \times \left(1 - \frac{1}{\beta^2 n^2}\right) \times \Delta E \times l$$

Other formulas to be considered:

- Raileigh scattering
- Dispersion formulas for different radiator material
- Reflection on spherical mirror
- Reconstruction of reconstruction angle by 4th order polynomial
- ….

3. Entities

Entities are usually derived from an analysis of the nouns describing the program. This approach we had followed earlier. In the second iteration we have seen how successful abstractions have been and the entities have been improved accordingly.

We want to point out that we describe here entities in the domain of RICH reconstruction. In a more general context a "Track" entity might have a slightly different meaning: It will be reduced to the basic features of a track. We do not think that the actual track entity in the future reconstruction program will be able to answer to RICH specific question. Only in the context of the RICH reconstruction program we are dealing with such an entity. (As in a different context the track entity could be extended in a different way). This concept is realized by extending the track entity by means of an adapter or bridge object.

We have noticed that the entities we found can easily classified according to their lifetime. Some will be instantiated at the start of the program and will be available during the execution of the whole program. Others will be instantiated for each event and will be available only during the processing of this event. And there is finally a class of objects that are available only temporary. Typically these objects contain the response to a complicated question asked to another entity.

3.1. With a lifetime over the whole job

Some entities represent detector elements and their functionality. Their lifetime extends over the execution of the whole program, that is the reconstruction of multiple events. Typically they will be instantiated at the initialization of the program and deleted at the finalization. During program execution they are providing functionality, such as photon generation or photon reconstruction, which are closely related to the detector. There will be a need (in the future) that these entities update their parameters from the detector description according to a database.

Obviously these classes are closely related to the detector hardware. We have found that it is more convenient to choose entities that are related to the function of the hardware (PhotonReflector) and not the concrete physical entity (as Mirror). The PhotonReflector entity provides an abstraction of the functionality provided. Obviously the concrete implementation has to know about the mirror (or the mirrors) involved in the photon reflection.

3.1.1. Rich

A container entity that is providing references to its components[2]. Any Rich entity might contain several PhotonRadiator entities, further a PhotonReflector and a PhotonDetector entity.

Questions

[2] We have to admit that the Rich entity is not a very strong entity. It hardly fulfills the criteria for being an independent object. Therefore it could be eliminated from the design. But to us it seems right to keep it as a container for its components.

- How many PhotonRadiators do you have?
- Give me a reference to PhotonRadiator i
- Give me a reference to your PhotonReflector
- Give me a reference to your PhotonDetector

In the following question concerning navigation in the class structure are suppressed.

3.1.2. PhotonRadiator

A PhotonRadiator Entity represents the photon emitting part of each Rich. In case of Rich 1 there is an aerogel radiator and a gas Radiator, in case of Rich 2 there is only a gas radiator. To provide answers to the following questions the radiator entity must know many aspects of the physics and the geometry of the radiator. But this is considered as an implementation detail that has no need to be visible in the list of questions. Care has been taken to ask questions in such a way that details, as the Rayleigh scattering in the aerogel radiator, are hidden from the user.

Questions

- Does a given TrackSegment enter the radiator? Where does it leave?
- What is the refractive index of the radiator material (at a given energy)?
- What is the momentum threshold for a given Particle Hypothesis?
- What is the saturated Cherenkov Angle?
- What is the emission spectrum (corrected for scattering) ?
- Generate for a given TrackSegment a random photon with given Particle Hypothesis
- Generate for a given TrackSegment a photon with given Cherenkov Angle and emission point.
- Reconstruct a photon for a given TrackSegment and a given Pixel
- What is the efficiency to observe a photon from a given TrackSegment for a given Particle Hypothesis. What fraction of the photons is scattered, which fraction is a true signal.

3.1.3. PhotonReflector

This entity represents the reflection system. It consists out of various mirrors.

Questions

- How is the photon spectrum modified by the mirror system ?
- Reflect a Photon. This is a complex question as the photon might also be absorbed.

- Reconstruct the photon. In case of the foreseen spherical mirror this question involves the solving of the 4th order polynomial.

3.1.4. PhotonDetector

It is representing the photon observation device. To describe its functionality it is assumed that a PixelID entity is describing a pixel position. This pixel position is associated with a position in a global coordinate system (simply a HepPoint3D). In addition a local coordinate system is provided, that allows for example to make easily a drawing of photon impact points at the detector plane in an event display.

Questions:

- Quantum efficiency for photon detection
- "Detect" a photon. Provides a PixelID. Keep in mind that a photon can also be absorbed.
- Convert PixelID, Local Position, GlobalPosition
- Area of a given pixel
- Number Noise Pixel to generate according to electronic noise
- Generate a noise pixel
- Noise Pixel to generate according to a particle transgressing the detector plane[3].
- Expected signal background for a pixel due to electronic noise

3.1.5. PhotoTube

This entity is planned for a future release. It is obvious that in a real detector a number of alignment and calibration constant have to be provided. The natural granularity will be the phototube. The photondetector contains all the tubes. After selecting the relevant tube, it will delegate the question.

3.2. With a lifetime spanning the reconstruction of one event

These entities describe event data. To support an easy and efficient evaluation of algorithms they provide also functionality that goes beyond that: e.g. a TrackSegment in a Radiator does not only know about its momentum or about extrapolations to various places, but also knows how many photons it would emit in this radiator. The actual calculation of the information is hidden from the user. It is delegated to the corresponding detector entity. In case the calculations are expensive, the event entities can also provide

[3] To be implemented in the future

some caching of these numbers. The cached results will disappear together with entity at the end of the processing of this event.

The entities as defined here should be useable both in the case of simulation and reconstruction. Most questions are important both for the simulation and the reconstruction. As explained later the actual implementation is naturally different. Some question will only be meaningful in one context

3.2.1. Event

A container object that provides reference to tracks and pixels

3.2.2. Track

The track as an entity per se. It contains track segments that do the real work. Some questions are valid on this level. In particular the track entity allows the setting of a hypothesis to support the reconstruction algorithm. If the hypothesis has been set for all tracks a “global hypothesis” has been chosen.

Questions

- Particle Momentum (at vertex)
- Particle Type (MC truth both for simulation and reconstruction)
- Charge
- Set/Get the Particle Hypothesis and corresponding Likelihood values
- Expected number of observed photons for the chosen hypothesis / for any given hypothesis (implies a loop on all segments)

3.2.3. TrackSegment

The track is passing a radiator. The TrackSegment entity describes the segment contained within that radiator. Consequently is has a length in the radiator and it can radiate photons along this path. To answer the questions this entity has an association with its PhotonRadiator

Question

- Particle Momentum
- Particle Direction at some extrapolation
- Particle Position at some extrapolation
- How many photons to generate with a given hypothesis

- Generate random photon with given particle hypothesis
- Reconstruct photon with given pixel
- Expected average Cherenkov Angle for given particle hypothesis
- Expected number of observed photons for a chosen hypothesis / for any given hypothesis
- Expected number of photons scattered / signal fraction
- Expected number of signal/scattered photons under a specific emission angle

3.2.4. Pixel

A hit pixel in the photon detector. Some questions depend on the global hypothesis set by the track entities. The Pixel has an association to its PhotonDetector to answer several questions.

Questions

- PixelID
- Signal
- Area
- Local Position
- Global Position
- Expected Background for this Pixel
- Expected Signal under the set global hypothesis

3.2.5. Photon

This entity represents the link between TrackSegment and Pixels. In case of the simulation they represent actual photons propagating through the detector. In case of the reconstruction it represents an hypothesis: It can provide answers to questions assuming that a particular pixel was illuminated by a given track.

Question

- What is the cherenkov angle

3.3. Which exists only temporary

Some entities exist only to provide answer to provide answers to question. Of particular importance are photon entities that represented simulation or reconstruction of photons. The question to these entities are different and it seems more clear to separate them.

Other entities are used to describe simple concepts like a PixelID. They can be part of the structural entities described before or used just as parameters to some question.

The list of entities is not complete and is meant to be illustrative.

3.3.1. PhotonSpectrum

Necessary to describe answers concerning the quantum efficiency of photon detectors, absorption in mirrors and emission spectra in radiator system. Implemented similar to a one-dimensional histogram.

Questions

- Content
- Integral
- Apply transformation
- Folding

3.3.2. ReconstructedPhoton

Represent a hypothetical Photon after reconstruction (which can also fail). Contains the information to create a Photon entity in the Event structure.

Question

- Status
- Emission Parameter
- PixelID

3.3.3. GeneratedPhoton

Represent a photon during and after generation process. This photon is generated in the radiator and afterwards propagated through the detector.

Question

- Status (Absorbed, Observed,Scattered)
- Emission Parameter (Angle, Energy)
- Current Position, current Direction
- PixelID (only if observed)

3.3.4. PixelID

A entity describing a pixel position.

Question

- Detector
- PhotoTube
- Pixel

4. Design

After the entities have been specified it is important to understand the necessary relations between the entities to implement their functionality. At this stage the design is shown only at an analysis level. This is sufficient to provide insight in the interplay between entities. For the actual implementation in C++ still a significant number of details have to be specified.

Please note also that at this level it is not necessary to provide details of the involved physics processes or calculations. They are understandable in general terms. These details are then implementation details. For an actual physics program they are naturally very important, because they have influence on the performance of the detector in real life.

4.1. Detector Configuration

In Figure 10 the relations between the detector entities is shown. This layer represents an interface that is specifying certain functionality. The actual implementation is hidden behind the base classes.

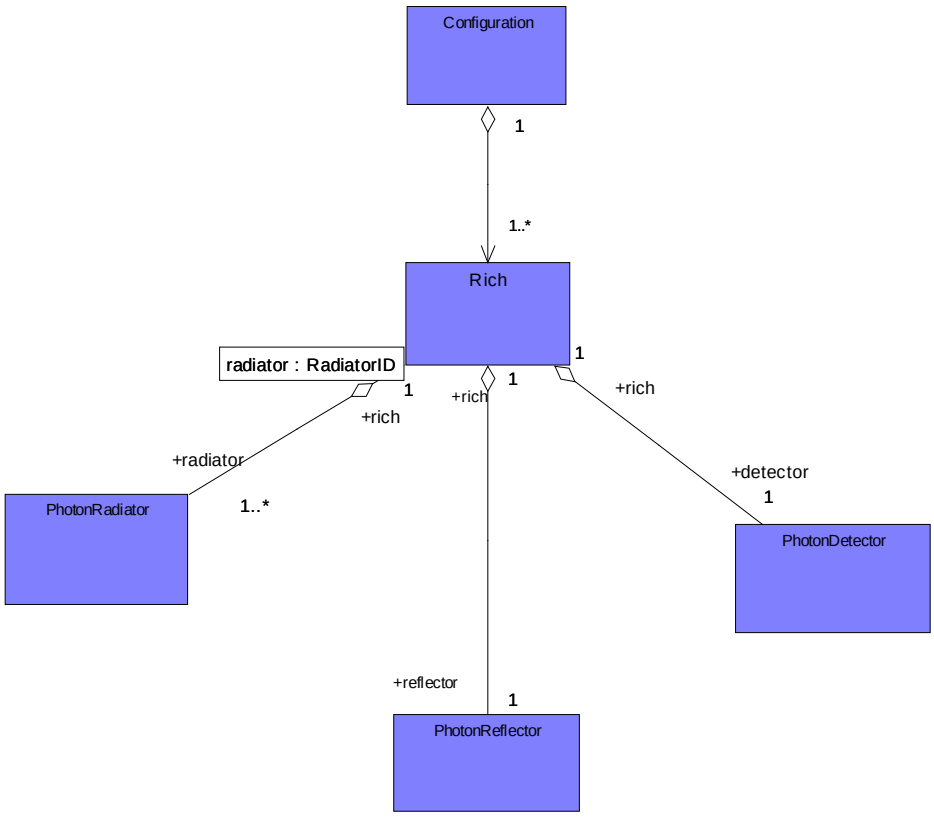

Figure 8:The detector components as UML Diagram

The configuration object is responsible for the allocation of the various detector components. For this purpose factory methods are provided. The current implementation is using an ASCII file to provide information on the detector geometry and the relevant detector parameters. In the future this will be replaced by an entity that is based on the object-oriented detector description and XML.

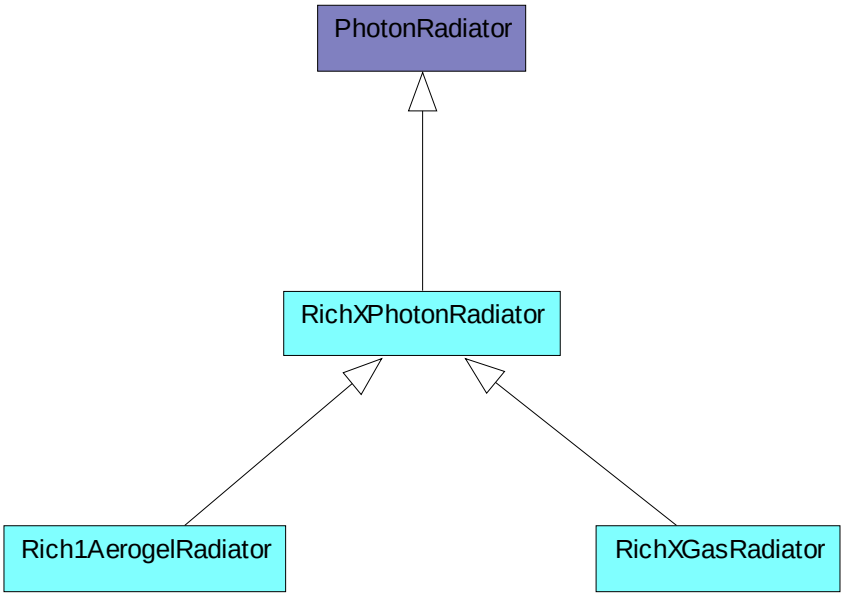

Figure 9: Inheritance tree of the Photon Radiator

As example how the implementation of the specific radiator components is implemented the inheritance tree of the photon radiator is shown in Figure 9. Rich1 has two photon-radiator systems – an aerogel radiator and a gas radiator. Rich 2 has only a gas radiator system. The gas radiator system are in principle very similar and are implemented in one class. The aerogel radiator is slightly different due to differences in the refractive index parameterization and the scattering in aerogel. Common functionality is implemented in a common intermediate class.

4.2. Event Model

In a similar way Figure 10 shows the relation between event model entities. Again this is just the definition of an interface and some common functionality.

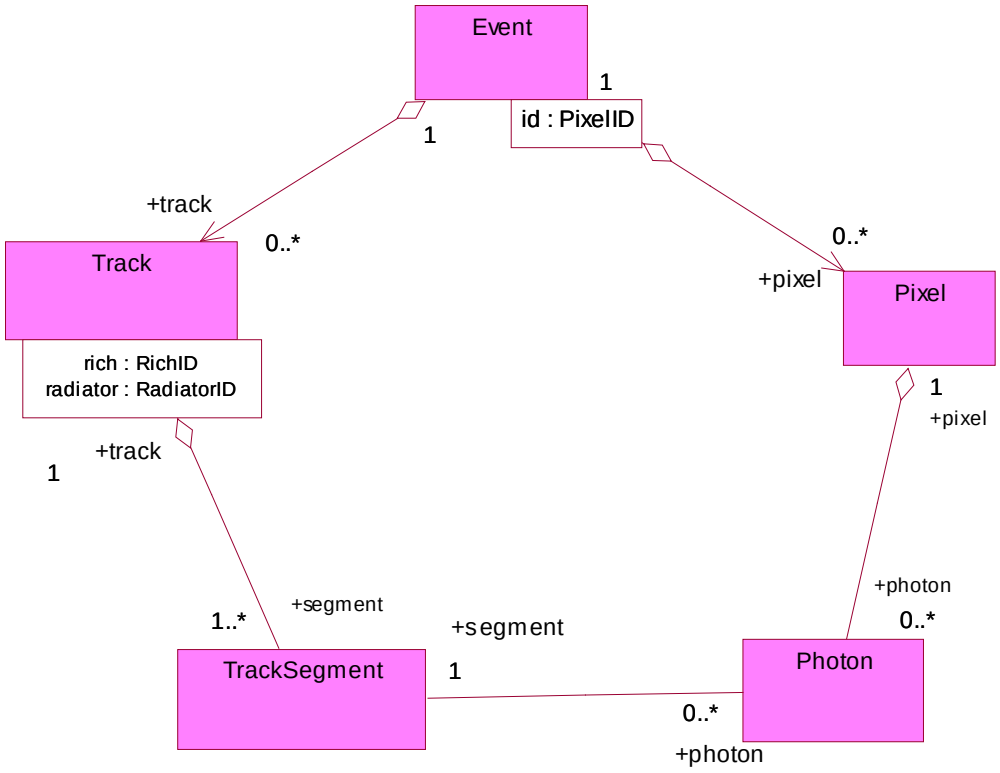

Figure 10:The event model

As shown in Figure 11 the true implementation is performed by an adapter. In most implementations the relevant information of the GAUDI track will be copied to the local entity to provide faster access. On the other hand this design allows to profit at a later stage from new features in the event model.

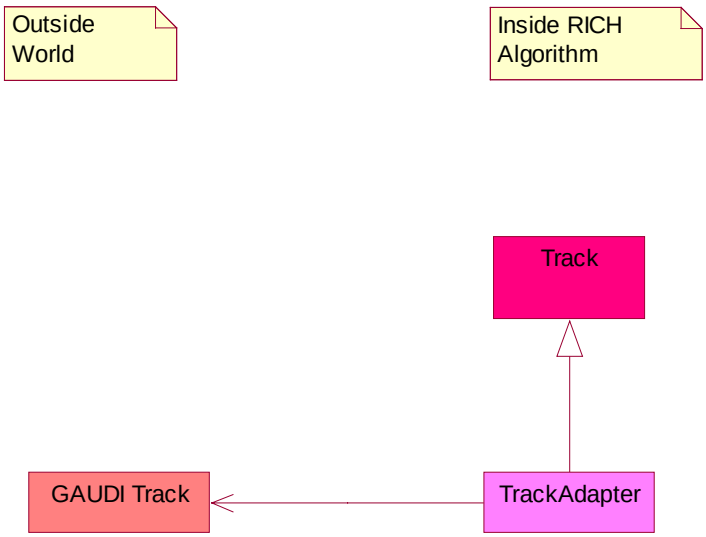

Figure 11: The track entity

The situation is more complicated if one is studying other entities, as the track segment in more detail. Figure 12 shows the track segemnt entity and its relation with other entities from the interface. In addition there is a relation between the track segment and the PhotonRadiator to allow the segment to answer to RICH specific questions. Natutally the actual implementaion of some features is again performed by an adapter

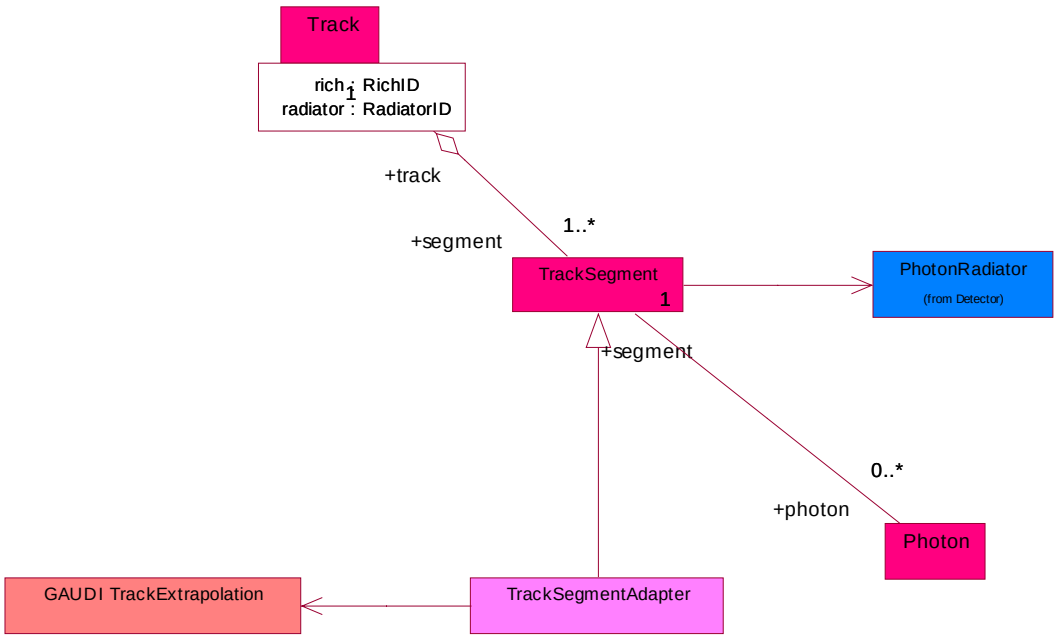

Figure 12:The track segment

5. Architecture

The architecture of an object-oriented system is concerned with the interplay of groups of entities. A number of classes are considered as a building block of a larger system. Interfaces are providing the glue between those blocks. We think that a reuse of components on algorithm level can be realized by composing the complete system out of building blocks. Such components are then elements that are useful in the context of Rich reconstruction programs[4].

We are thinking here of components for the actual reconstruction program, a possible trigger program, a fast simulation program[5], but also in a alignment and calibration problem. A further advantage of components is that they are providing a mean to perform updates of the software in an incremental way: A new detector implementation might not be used in production immediately. Or several components could be offered which provide a different implementation for comparison purposes. Well defined interfaces de-couple the component from the other parts of the program

As mechanism for the assembly of the components GAUDI sub-algorithms seem to be the ideal choice. Firstly they provide a standard way to create instances via a factory method. A number of standard components will be provided in the shared library. They are instantiated according to parameters of the JobOption file. This allows the user also to create his own variant that can be loaded at run time instead of the standard version. Another strong point for this choice is that a sub-algorithm has also easy access to GAUDI services, which is of obvious advantage.

[4] It might be interesting to consider if some functionality could be provided to users outside the Rich. If such a component can be identified it would be also possible to provide it as global service or tool.

[5] The full simulation will be probably be performed in Geant4 – including the photon simulation. It has to be seen at which point the current approach for simulation photons can be replaced. The timescale for this is maybe quite long.

5.1. General Outline

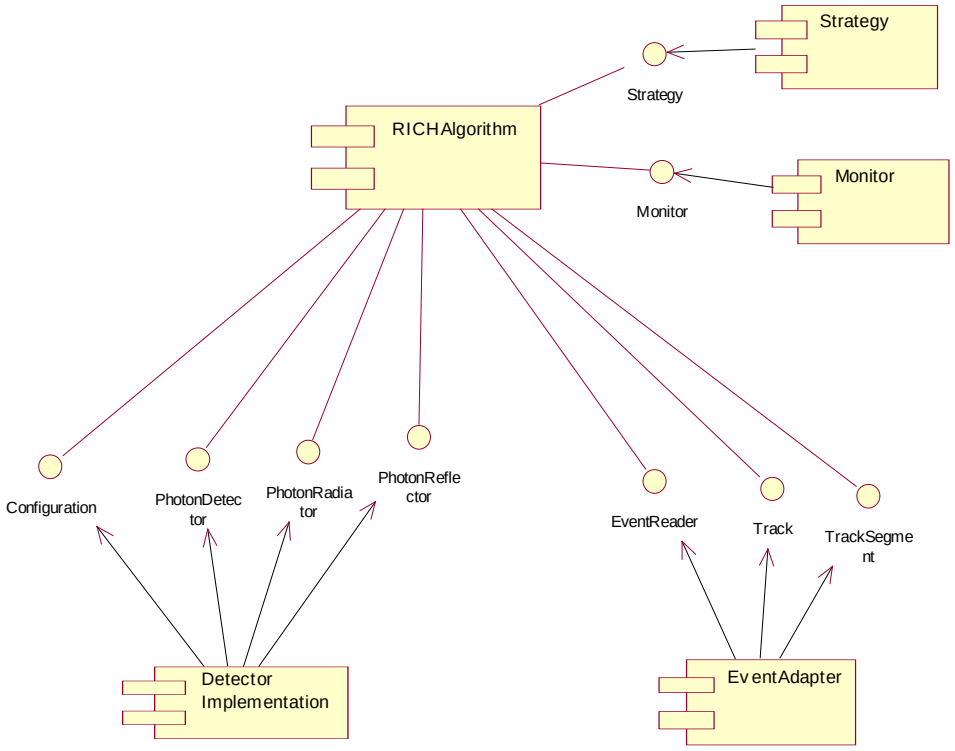

Figure 13: RICH algorithm component diagram

5.1.1. Detector Configuration

The detector configuration is the most important building block. It provides the implementation of all fundamental physics routines for photon simulation and photon reconstruction. The configuration object itself is derived from a GAUDI algorithm. At program start the RICH Algorithm creates an instance of the configuration object.

On its initialization it creates then the already well known tree of detector functionality components (see Figure 8). During execution of the event analysis it is providing its entities to the algorithm to do the actual work of the simulation or reconstruction. Different versions are possible to implement different detector variants. In addition different versions of the implementation could be provided.

In the next future two versions are foreseen: A version derived from the current implementation, where the geometry is taken from an ASCII file. And a future version, where the information is extracted from the object oriented detector description.

5.1.2. Event Adapter

Event adapters need a different strategy. The RICH Algorithm is creating an instance of an EventReader. When a new event has to be analyzed the algorithm requests the next event from the reader. It establishes contact with the data store and creates instances of the various Track and TrackSegment adapters. It then provides these entities to the algorithm to let the algorithm do its work. At the end of the processing the event tree is again destroyed. Now result data will be entered into the data store.

An obvious difference to the detector implementation is that for different purposes different event adapters are necessary. For example the Simulation Event Adapter will access MC information and it will finally deposit RICH raw data in the event store. A reconstruction adapter will access reconstructed track data and detector raw data. It will provide at the end detector reconstruction data. Currently Monte Carlo information is used to fake raw data.

Various versions of these event adapters will be provided:

- For the simulation
- For the reconstruction using simulation tracks assuming ideal track reconstruction
- For the reconstruction using simulation tracks simulation inefficiency
- For the reconstruction using reconstruction tracks

Obviously there are also different versions imaginable depending on the progress of the work.

5.1.3. Strategy

The RICH algorithm does not know by itself about its tasks: For that purpose a Strategy object is provided. It follows the standard Strategy Pattern [6]. For the evaluation of an event the Algorithm object is providing the context, e.g. the detector configuration and the event. Put in more simple turn the algorithm object is calling an execute methode of the a strategy object, e.g.

```
strategy->execute(event,configuration);
```

The strategy object is then performing its task and modifying the event model. At the end the control is returned to the main algorithm object, that will perform the cleanup and transfer the results in the data store.

Again various versions of these object are imaginable

- For simulation
- For reconstruction using the global likelihood algorithm (in different development stages)
- For reconstruction using different algorithms
- At a later point also different tasks such as alignment or calibration will be added

5.1.4. Monitor

The monitor objects are conceptually very close to the strategy object. The main difference is that they have only read access to the event model, e.g. they cannot provide results in terms of modification of the event model. But they can fill histograms, log files etc. In principle any number of monitor objects can be attached to an algorithm.

Monitor can be also very useful for the user of the reconstruction program. He can use standard monitors provided in the implementation. It will be also straightforward to create own monitor objects that monitor different aspects of the system.

A number of monitor objects will be provided for various purposes. Right now a Simulation Monitor, a Simulation Event Display and a Reconstruction Monitor are available. Examples will be provided how to create your own monitor.

6. Conclusions and Outlook

In the document an object-oriented system for RICH reconstruction has been described. It is well adapted to the GAUDI environment and profits from its features.

A first implementation of the Algorithm has been obtained in C++ and it has been found compatible with the previous algorithm both in physics performance and in resource consumption.

For the future a more formal way of decomposing the algorithms into components is forseen. It will allow the assembly of algorithm at run time. This has not only advantages for the maintainability of the program, but it offers also practical way to reuse components for different purposes.

In the ongoing development we plan to progress both in terms of structure of the program and in the implementation of new features.

7. References

[1] LHCb Technical Proposal Chapter 9, RICH Detectors (1999)

[2] R. Forty and O. Schneider, RICH Pattern Recognition, LHCb-Note 98-040, RICH

[3] A. Schöning, Particle Identification in the RICH detectors and study of impact parameters, LHCb-Note 97-018, RICH

[4] D. Liko and N. Neufeld, Presentation on the RICH Meeting, November 1999 http://lhcb.cern.ch/rich/powerpoint/oorich_liko.ppt

[5] Kuck & Associates, http://www.kai.com

[6] Gamma et al., Design Pattern, Addison-Wesley