

Object Databases in the LHCb¹ Experiment

R. Chytrcek

CERN, Geneva, Switzerland, Radovan.Chytrcek@cern.ch

Abstract

Database systems are very important in any HEP² experiment in the world. HEP community in LHC³ era is in transition from FORTRAN to C++ and from data streams to persistent objects. Together with that a new data management will be necessary, which would allow transition from “files and tapes” approach towards the access to data in form of objects by selection of their required physics contents. In order to conform to the object-oriented paradigm, LHCb has to heavily investigate in the design and development of object databases for both the on-line (data acquisition and real-time processing) and off-line (simulation, reconstruction and analysis) computing environments, e.g. the Event Store, Detector Description Database, Calibration and Alignment Database etc. The focus of this article is on the Gaudi[2][3] framework and its very important part the detector description database.

Keywords: ODBMS, HEP, detector description database, object oriented, architecture, framework, component, persistency

1 Computing background of the LHCb experiment[1]

A data-flow analysis has been made of the computing system, in order to identify the key tasks and data flows. Estimates of computing requirements are made by analyzing existing LHCb simulation, reconstruction and analysis programs, and, in cases where they do not yet exist, by extrapolations from similar codes developed by other experiments. Results show that the total CPU requirements needed for simulation, for Level-2 and Level-3 triggers, and for reconstruction and analysis of simulated and real data, amounts to $\sim 5 \times 10^6$ MIPS. The raw data is written to permanent storage at a rate of 20 MB/s. Expected raw data volume and rates are show in the Table 1.

Table 1: Raw data volumes and rates of the LHC experiments[4]

TIME INTERVAL	ATLAS & CMS	ALICE	LHCb
	1 MB/event, 100 Hz	40 MB/event, 40 Hz	100 KB/event, 200 Hz
1 second	100 MB	1,6 GB	20 MB
1 minute	6 GB	100 GB	1,2 GB
1 hour	360 GB	6 TB	75 GB
1 day	8,6 TB	140 TB	1,7 TB
1 year ^a	1 PB	1 PB	0,7 PB

a. means the amount of useful information collected during 1 year

¹ Large Hadron Collider Beauty experiment (precision measurements of CP-Violation and rare decays)

² High Energy Physics

³ Large Hadron Collider

Attention will be given to the quality of software, particularly for the high-level trigger algorithms. The extra effort that this implies can be recovered by extracting as much use as possible out of each software component by ensuring its reuse in all application domains. This implies investment in sound engineering and management practices. The LHCb simulation program is written in FORTRAN, but are being re-engineered using Object Technologies.

Instead of development of a new ODBMS⁴ the strong emphasis is put on design of the information systems which can exploit existing ODBMS technologies, especially object persistency, but on the other hand should provide programming interfaces, frameworks and tools for physicists to satisfy their needs and requirements.

2 Data processing

A software system should cover all the tasks needed for processing of the event data. It spans the domains of on-line and off-line computing. It includes the algorithms to filter interesting events from background (so called high level triggers), as well as the full reconstruction and analysis tasks. The overview of these tasks are shown in Figure 1.

The data processing system consists of a series of processes or tasks that will transform the data collected from the detector into physics results. The data processing is done in various stages following the steps: data collection and filtering, physics and detector simulation, reconstruction and finally physics analysis. The development of all these data processing tasks involves in one hand computing specialists to build the framework and the basic components and the other hand the people specialized on each sub-detector that will build the specific code which will be needed for reconstruction, simulation of each of the sub-detectors and its final combination. In addition, there are people developing the data analysis programs to produce the final physics results.

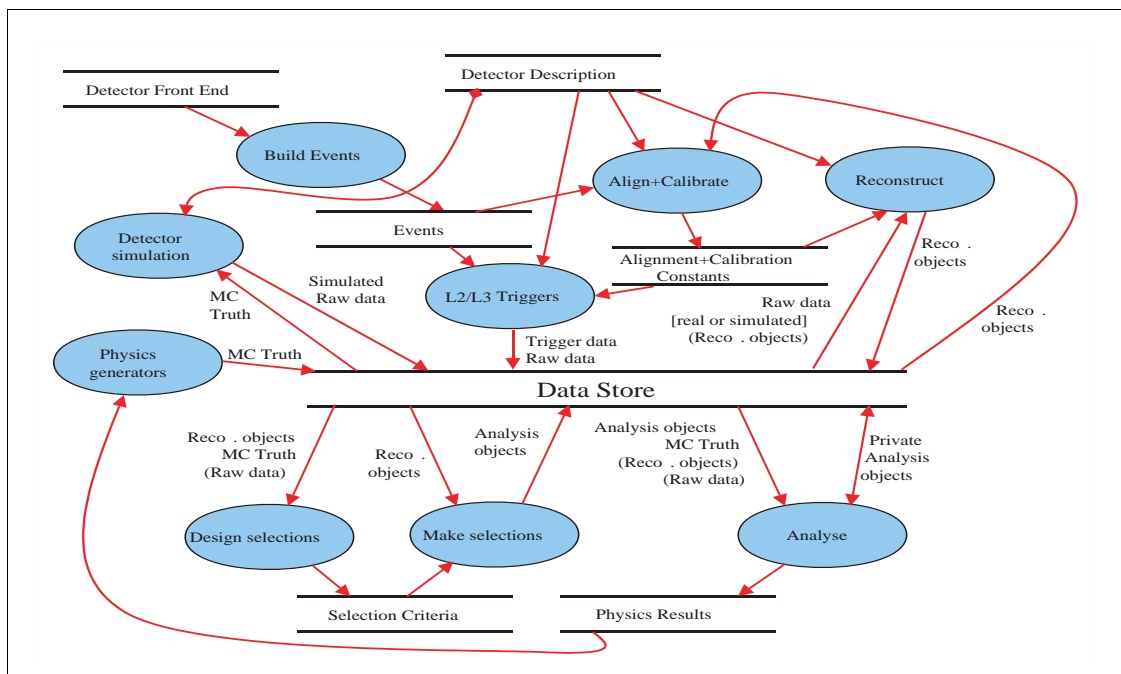


Figure 1: Dataflow diagram showing all the main tasks and data flows for the LHCb computing[1]

⁴ Object Database Management System

3 Gaudi framework

The Gaudi project has been started in order to fulfill all the needs and requirements for data processing. Its goal is to build a framework⁵ which can be applied to a wide range of physics data processing applications for the LHCb experiment. It should cover all stages of the physics data processing and wide range of different environments such as interactive and batch applications, on-line, off-line etc.

The framework is the implementation of the architecture⁶, see Section 4. So, it is important that the framework has sufficient knobs, slots and tabs that can be adapted to current problem and integrated to other frameworks. To understand the Gaudi architecture and its design it is needed to describe the major design criteria and strategic decisions that has been taken.

3.1 Clear separation between “data” and “algorithms”

Despite the intention to produce an object oriented design, it has been decided to separate data from algorithms. For example, the idea is to have “hits” and “tracks” as basically *data objects* and to have the *algorithms* that manipulate these data objects encapsulated in different objects such as “track_fitter” or “cluster_finder”. The methods in the *data objects* will be limited to manipulations of internal data members. An *algorithm* will, in general, process *data objects* of some type and produce new *data objects* of a different type. For example, the cluster finder algorithm, produces cluster objects from raw data objects.

3.2 Three basic categories of data: event, detector and statistical data

There are three major categories of *data objects*. There will be the *event data* which is the data obtained from particle collisions and its subsequent refinements (raw data, reconstructed data, analysis data, etc.). Then, there will be *detector data* which is all the data needed to describe and qualify the detecting apparatus in order to interpret the *event data* (structure, geometry, calibration, alignment, environmental parameters, etc.). And finally, there are *statistical data* which will be the result of some processing applied to a set of events (histograms, n-tuples, etc.), see Table 2.

Table 2: LHCb data volumes[5]

	Real (KB/event)	Simulated (KB/event)	Real	Simulated	Total (TB/year)
Raw data	100	200	200	200	400
Reconstruction	70	150	140	150	290
Analysis	< 10	< 10	~1	~1	~2
Secondary data (calibration and alignment, geometry and other databases, metadata, etc.)					~2
Total			~340	~350	~700

⁵ A framework represents a collection of classes that provide a set of services for a particular domain

⁶ The software architecture of a program or computing system is the structure or structures of the system, which comprises software components, the externally visible properties of those components, and relationships among them

3.3 Clear separation between “persistent data” and “transient data”

A main feature of the design is the separation of the persistent data from the transient data for all data types e.g. event, detector description, histograms, etc. The idea behind is that physics algorithms should not use directly the data objects in the persistency cache but instead use pure transient objects. There are several reasons for that choice:

- The majority of the physics related code will be independent of the technology for object persistency.
- The optimization criteria for persistent and transient storage are very different. In the persistent world you want to optimize I/O performance, data size, avoid data duplication to avoid inconsistencies, etc. On the other hand, in the transient world you want to optimize execution performance, ease of use, etc. Additionally you can afford data duplication if that helps in the performance and ease of use.
- To plug existing external components into the architecture they have to be interfaced to data. If they are interfaced to the transient data model, then the investment can be reused in many different types of applications requiring or not requiring persistency. In particular, the transient data can be used as a bridge between two independent components.

3.4 Data centered architectural style

The architecture should allow the development of *physics algorithms* in a fairly independent way. Since many developers will be collaborating in the experiment software effort, the coupling between independent algorithms should be minimized. It has been envisaged using a *transient data storage* as a means of communication between algorithms. Some algorithms will be “producing” new data objects in the data store whereas others will be “consuming” them. In order for this to work, the newly produced data objects need to be “registered” somehow into the data store such that the other algorithms may have the possibility of identifying them by some “logical” addressing schema.

3.5 Encapsulated “User code” localized in few specific places “Algorithms” and “Converters”

It is needed to take into account the need to customize the framework when it is used by different event data processing applications in various environments. Most of the time this customizing of the framework will be in terms of new specific code and new data structures. Therefore it is needed to create a number of “place holders” where the physics and sub-detector specific code will be later added. Two main places are considered: *Algorithms* and *Converters*.

3.6 All components with well defined “interfaces” and as “generic” as possible

Each component of the architecture implements a number of interfaces (pure abstract classes in C++) used for interacting with the other components. Each interface consists of a set of functions which are specialized for some type of interaction. The intention is to define these interfaces in a way as generic as possible. That is, they should be independent of the actual implementation of the components and also of the concrete data types that will be added by the users when customizing the framework.

3.7 Re-use standard components wherever possible

It is intended to have one single team with an overview of the complete LHCb software system covering the traditional domains of off-line and on-line computing. The hope is in this way to avoid unnecessary duplication by identifying components in the different parts of the system which are the same or very similar. All is going towards the re-use of standard and existing components wherever possible.

3.8 Integration technology standards

Currently it is not a proper time to select a standard integration technology that provides the glue between the different components of the architecture. However, we are aware of the importance of selecting a technology and standardizing on it to guarantee a smooth integration and facilitate re-use of commercial components. For this reason, we have tried to follow some of the ideas behind the more popular integration technologies (CORBA, DCOM, JavaBeans) such that their later adoption will not be traumatic.

4 Gaudi Architecture

A few terms and parts of Gaudi architecture were mentioned already in the last chapter. Now the description of the architecture will be introduced, as shown in the Figure 2. Maybe the object diagrams are not the best way to show the structure of the software but they are very illustrative for explaining how the system is decomposed. They represent a hypothetical snapshot of the state of the system, showing the objects (in this case component instances) and their relationships in terms of navigability and usage.

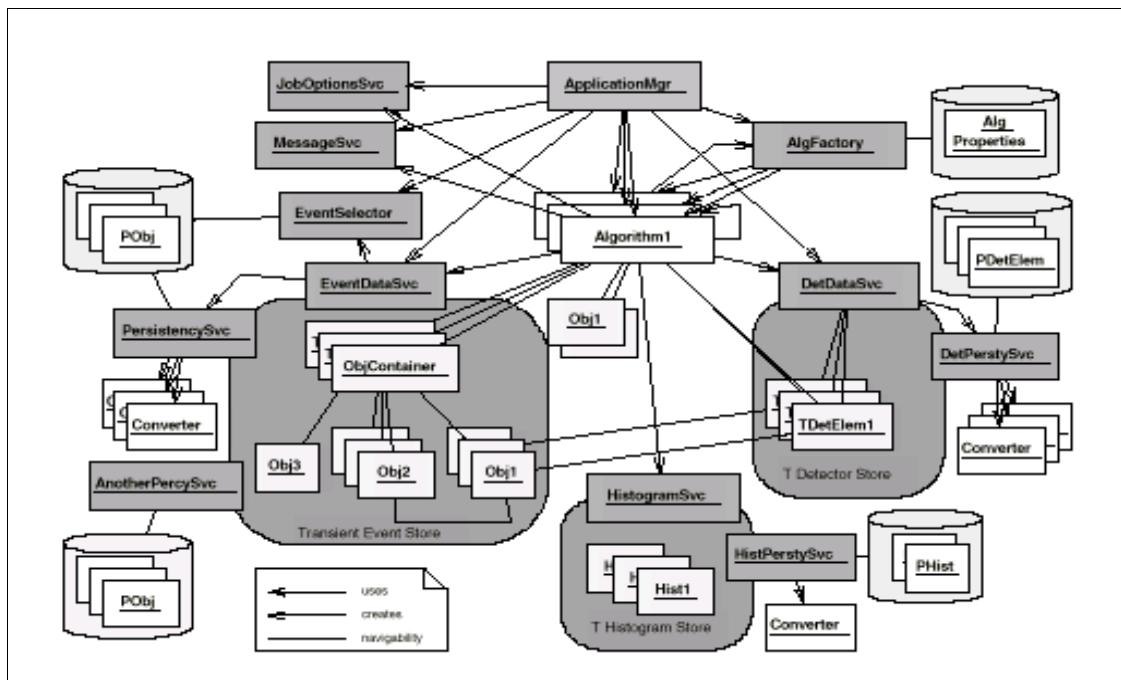


Figure 2: Object diagram of the architecture covering the kernel of any data processing application

4.1 Algorithms and Application Manager

The essence of the event data processing applications are the physics algorithms. They are encapsulated into a set of components called *algorithms*. These components implement a standard set of generic interfaces. *Algorithms* can be called without knowing what they really do. In fact, a complex *algorithm* can be implemented by using a set of simpler ones. At the top of the hierarchy of *algorithms* sits the *application manager*. The *application manager* is the “chef d’orchestre”, it decides what algorithms to create, when and in which order to call them.

4.2 Transient data stores

The data objects needed by the *algorithms* are organized in various *transient data stores*. The data are distributed over three stores as shown in the Figure 2. This distribution is based on the nature of the data itself and its lifetime. The event data which is only valid during the time it takes to process one event is organized in the *transient event store*. The detector data which includes the detector description, geometry, calibration, etc. and generally has a lifetime of many events is stored in the *transient detector store*. Finally, the statistical data consisting of histograms and n-tuples which generally have a lifetime of the complete job is stored in the *transient histogram store*. It is understood that the three stores behave slightly differently, at least with respect to the data lifetime (the event data store is cleared for each event), but their implementations have many things in common. They could be simply different instances of a common component.

4.3 Services

A number of components have been defined which should offer all the services directly or indirectly needed by the *algorithms*. The idea here is that it would be nice to offer high level services to the physicist, so that they can concentrate on developing the physics content of the algorithms and not on the technicalities needed for making everything work. This category of components is called *services*. Some examples of services can be seen in the object diagram. For instance, there are services for managing the different transient stores (*event data service*, *detector data service*,...). These services should offer simplified data access to the algorithms. Another class of service are the different *persistency services*. They provide the functionality needed to populate the transient data stores from persistent data and vice versa. Other services like the *job options service*, *message service*, *algorithm factory*, etc. which are also shown in the diagram offer the service which its name indicates.

4.4 Converters

The converters help the persistency services to populate the transient data store with specific data types. They know how to convert a specific data object from its persistent representation into its transient one or the other way around.

4.5 Selectors

A number of components has been envisaged whose function will be to select. For instance, the *event selector* provides functionality to the *application manager* for selecting the events that the application will process. Other types of selectors will permit choosing what objects in the transient store are to be used by an algorithm, by a service, etc.

5 Detector Description Database (DDDB)

The scope of the DDDB is shown in the Figure 3. Detector description database contains identifiable entities and should provide an information about logical detector hierarchy, its subsystems and many links, e.g. detector geometry parameters, conditions (alignment, calibration, slow control), electronics channels mappings and magnetic field. The navigation should be provided here allowing to browse and select data in DDDB. Navigability in the detector description is enabled by logical tree like organization of identifiable detector elements thus reflecting the detector logical structure in a natural way reflecting the reality.

Selection of detector description data will be done using metadata thus avoiding the access to the real persistent data. The term metadata is used here in sense that metadata define structure and relationships among the data in the DDDB persistent store. The metadata goes into persistent data store together with other detector data but its amount is much smaller. Thus they can be loaded into transient store and provide the interface to access the real detector data without searching the huge persistent data store. As they describe the relationships among the various detector data, the metadata can be used to build the logical organization of the transient detector description store and help to resolve detector data associations in run-time.

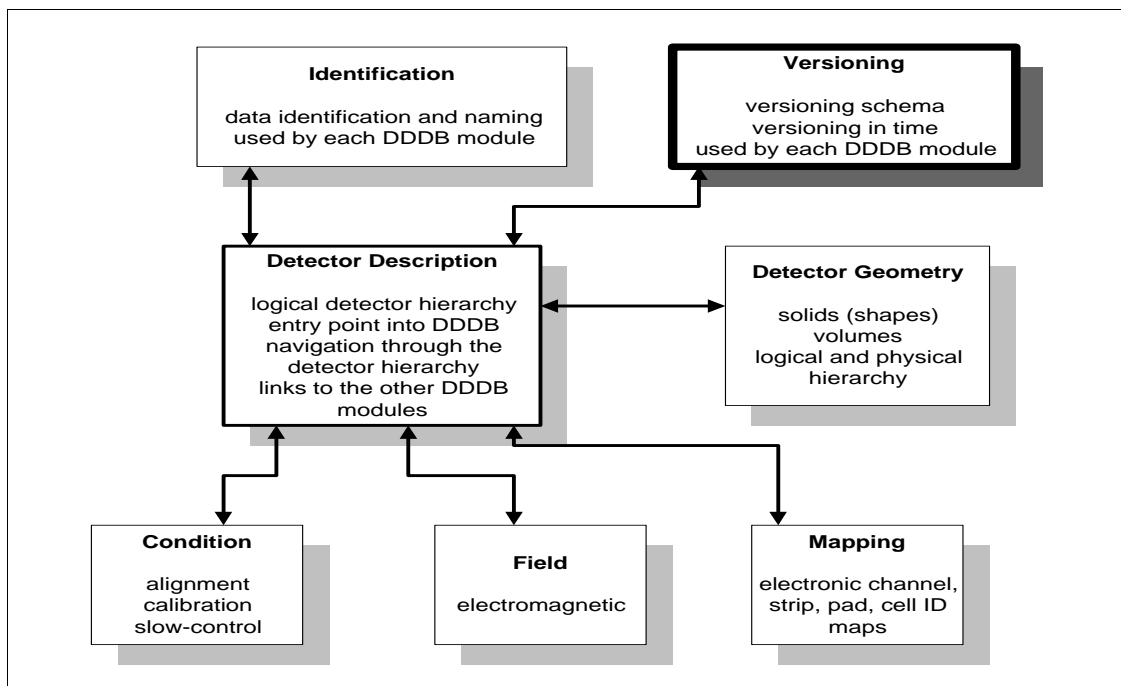


Figure 3: DDDB scope

One of the most important requirements on DDDB is the capability to deal with multiple versions of the detector data. There are versions evolving in time. The detector data are time stamped according to the time they were measured or time when the event was taken. In addition to that the sets of conditions constants will be usually refined after each data reprocessing (possibly several times a year), which adds second dimension into the versions schema, as there is a possibility to have more than one version of the detector geometry valid in the same validity range. The right version is selected by the user application according to its needs and purpose. It does not mean that the latest version is the right one. The physics applications have different needs so it is very important to provide logical views to the

detector geometry. For example a fast simulation program requires simple detector geometry description but detailed simulation needs very detailed information about the simulated detector element. As there will be only single DDDDB, to select the right level of details together with required version of conditions data is non-trivial task.

6 DDDDB Design

6.1 DDDDB Transient model

Transient model of the detector description has been designed and its initial prototype implemented. The part of the model of the transient detector description store is shown in the Figure 4. This model reflects how the detector elements and geometry data are dealt with in reality. Objects of DetectorElement class describe logical detector hierarchy. Each detector elements provides an interface to all of the relevant information, and container of its descendants in the hierarchy.

Objects of the PhysicalVolume class hold information about position in space and information about the logical volume representing geometry parameters and the parent logical volume what means where the physical volume is placed in the geometry hierarchy. More than one physical volume object can point to the same logical volume. This allows to handle multiple replicas of the detector elements which have the same volume properties, but vary in position and transformation.

LogicalVolume class objects define geometry properties of the detector elements, e.g. volume shape, dimensions, material and, what is very important, its daughter volumes in geometry hierarchy.

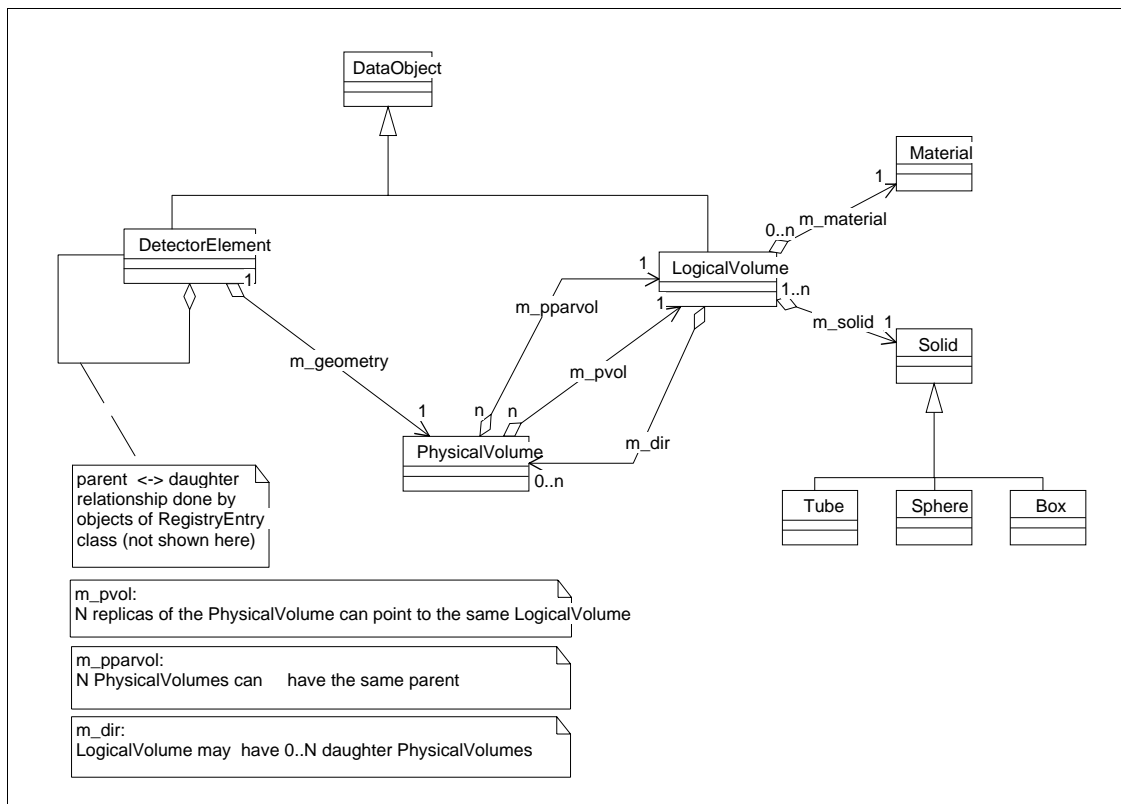


Figure 4: Partial design of the transient detector description model

This model allows navigation in detector elements hierarchy to find and identify the information about a given detector element. It is meant to be the user entry point into DDDB. PhysicalVolume class objects provide the bridge between logical detector description and geometry detector description. Navigation in the geometry hierarchy is done by traversing the geometry tree alternating from physical to logical volumes and vice versa.

6.2 DDDB Prototype

The prototype is capable to build the logical detector description structure in the transient store using identifiable elements and resolve links to related geometry data. The main goal is to define LHCb detector description including its basic geometry hierarchy. This involves some work on converters which can translate information about detector geometry, currently stored in text files, into transient representation. In parallel with that, the simple metadata system has been implemented, which allows to retrieve the relevant information for Gaudi detector data service. With this information the data service can build the structure in the transient store and resolve the links between the detector elements and their geometry representation. After having the detector logical structure description built it is possible to provide concrete converters for each of the subdetectors and convert the geometry information provided by LHCb sub-detector groups.

7 Conclusions

At the first look, model of DDDB transient store may look simple, but the model reflects many requirements of different physics applications (simulations, reconstruction and analysis) and takes into account the interoperability with other HEP software packages (especially conversions of detector geometry data). The model defines data structures that are easy to use in a way natural to physicists. Regardless of its relatively simple design the model should be able to handle detector data complexity arising from huge amount of detector elements and geometry volumes. Very much work has been done on analysis, use cases and requirements. The resulting model has incorporated the collected knowledge or at least major part of it into its design.

DDDB Prototype has been integrated within the Gaudi framework. Even the functionality it offers is far from the requirements it allows already to work with basic logical detector description which can be used by the physics applications based on Gaudi framework.

References

- 1 LHCb Collaboration, *LHCb Technical Proposal*, European Laboratory for Particle Physics (CERN), CH-1211, Genève 23 - Suisse, ISBN 92-9083-123-5, 1998
- 2 P. Mato and LHCb software architecture group, *Gaudi - Architecture Design Document*, LHCb experiment, European Laboratory for Particle Physics (CERN), CH-1211, Genève 23 - Suisse, LHCb/98-064 COMP
- 3 P. Maley and LHCb software architecture group, *Gaudi User Guide*, LHCb Experiment, European Laboratory for Particle Physics (CERN), CH-1211, Genève 23 - Suisse, 1999
- 4 P. Binko, *Object Oriented Databases in High Energy Physics*, in Proceedings of CERN SCHOOL OF COMPUTING 1997, ISBN 92-9083-120-5
- 5 P. Binko, *LHCb Computing Tasks*, LHCb experiment, European Laboratory for Particle Physics (CERN), CH-1211, Genève 23 - Suisse, LHCb/98-042 COMP