Software for the LHCb Experiment

Gloria Corti, Marco Cattaneo, Philippe Charpentier, Markus Frank, Patrick Koppenburg, Pere Mato, Florence Ranjard, Stefan Roiser, Ivan Belyaev, and Guy Barrand

***Abstract*—LHCb is an experiment for precision measurements of CP-violation and rare decays in B mesons at the LHC collider at CERN. The LHCb software development strategy follows an architecture-centric approach as a way of creating a resilient software framework that can withstand changes in requirements and technology over the expected long lifetime of the experiment. The software architecture, called GAUDI, supports event data processing applications that run in different processing environments ranging from the real-time high-level triggers in the online system to the final physics analysis performed by more than 100 physicists. The major architectural design choices and the arguments that lead to these choices will be outlined. Object oriented technologies have been used throughout. Initially developed for the LHCb experiment, GAUDI has been adopted and extended by other experiments. Several iterations of the GAUDI software framework have been released and are now being used routinely by the physicists of the LHCb collaboration to facilitate their development of data selection algorithms. The LHCb reconstruction (Brunel), the digitization (Boole) and analysis (DaVinci) applications together with the simulation application (Gauss), also based on Geant4, and event and detector visualization program (Panoramix) are all based on the GAUDI framework. All these applications are now in production.**

***Index Terms*—Programming, software, software packages.**

I. Introduction

THIS paper presents the current status of the LHCb software. LHCb [1], [2] is one of the experiments currently under construction at CERN to take data at the Large Hadron Collider (LHC). The primary goal of the experiment is to make precision measurements of CP violating decays and other rare phenomena in the b-system and so to make detailed tests of the Standard Model description of CP violation, as well as investigate possible new physics. LHCb will produce large amounts of data, of the order of Peta bytes per year, which will need to be reconstructed and analyzed to produce the final physics results. In addition, physicists are continuously studying the detector and the physics performance that can be achieved using it. Software for all data processing stages for the various needs of the experiment has been produced and is at different levels of development. This software will have to be maintained throughout the lifetime of LHCb, expected to be of the order of 10–20 years; the impact of changes in software requirements and in the technologies used to build software can be minimized by developing flexible and adaptable software that can withstand these changes and can be easily maintained over the long timescale involved.

With these goals in mind we have constructed GAUDI [3], a general object oriented framework designed to provide a common infrastructure and environment for the different software applications of the experiment. The applications, supporting the typical phases of Particle Physics experiments software, from simulation to reconstruction and analysis, are built within the GAUDI framework. Experiment specific software, as for example the Event Model and Detector Description, are also provided with the framework as core software components. The framework together with these services and the applications constitutes the complete LHCb software system. The subdetector software developers, or physicists performing analysis, provide the software algorithms to these applications. Use of the framework in all applications helps to ensure the integrity of the overall software design and results in maximum reuse of the core software components.

Manuscript received November 15, 2004.

G. Corti, M. Cattaneo, Ph. Charpentier, M. Frank, P. Koppenburg, P. Mato, F. Ranjard, and S. Roiser are with the PH Department, CERN, CH-1211 Geneva 23, Switzerland (e-mail: gloria.corti@cern.ch).

I. Belyaev is with the Laboratoire d'Annecy-Le-Vieux de Physique des Particules, BP 110, Chemin de Bellevue, F-74941 Annecy-le-Vieux Cedex, France, on leave from the Institute for Theoretical and Experimental Physics, Moscow, Russia.

G. Barrand is with the Laboratoire de l'Accelerateur Lineaire, Centre d'Orsay, Bat. 200, F-91898 Orsay, France.

Digital Object Identifier 10.1109/TNS.2006.872627

II. GAUDI Architecture and Framework

The development process for GAUDI is architecture centric, requirements driven, incremental, and iterative. This involves identifying components with specific functionality and well-specified interfaces, defining how they interact with each other to provide the whole functionality of the framework. The framework is real code implementing the building blocks outlined in the architecture and ensuring its design features are respected. The approach to the final software system is via incremental releases, adding to the functionality at each release according to the feedback and priorities of the physicists developing the code for the different applications and following the evolution and changes in their needs.

A schematic view of the GAUDI architecture can be seen in the object diagram shown in Fig. 1. It represents a hypothetical snapshot of the state of the system showing the objects (in this case component instances) and their relationships in terms of ownership and usage. Note that it does not illustrate the structure of the software in term of class hierarchy. In the following we will outline the major design choices taken in the GAUDI architecture.

A. Generic Component Model With Well-Defined Interfaces

Each component of the architecture implements a number of interfaces (pure abstract classes in C++, the main language used in the implementation) for interacting with the other components. The basic idea of GAUDI is to define a basic set of services that are common to most of the event data processing

0018-9499/$20.00 © 2006 IEEE

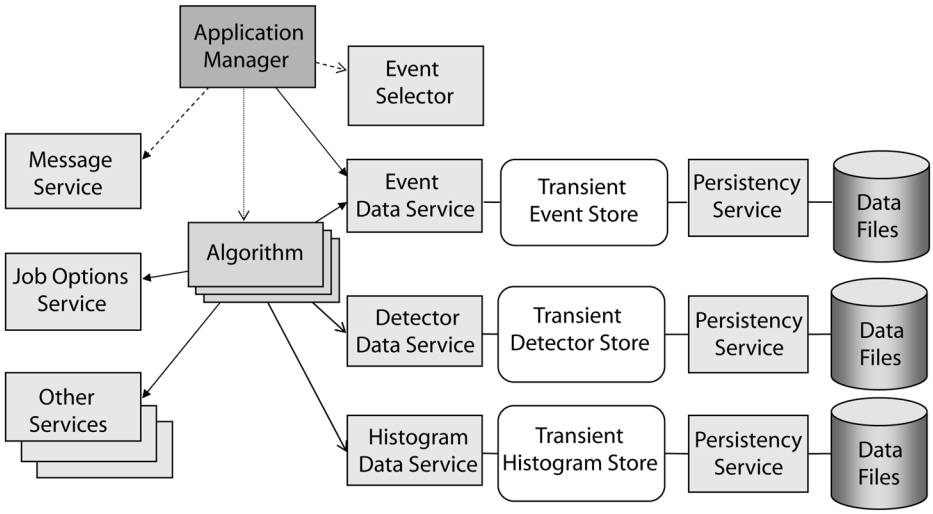

Fig. 1. Object diagram of the GAUDI architecture.

applications that LHCb had to develop and to define their interfaces independently of the actual implementation. In order to ease the integration of components we defined an interface model supporting interface versioning, dynamic interface discovery, and generic component factories. With these features we were able to implement runtime loading of components (dynamic libraries) allowing us to use a plug-and-play mechanism in the implementation of the data processing applications. Since all components are essentially decoupled from each other, they can be implemented independently and in a minimal manner, i.e., supplying sufficient functionality to do their job but without the many refinements that can be added at a later date. Components can be developed using other specialized frameworks or toolkits, for example for data persistency, visualization, simulation, etc. A specific implementation of a component can be replaced by another one implementing the appropriate interface and providing equivalent functionality. This makes possible a transparent use of *third-party* software. This approach has allowed us to build the LHCb applications by customizing the framework, i.e., by dynamically selecting the most suitable components to perform the different tasks. Due to these features the GAUDI framework is easily adaptable for use in other experiments: although originally developed for the LHCb experiment it has been adopted and extended by the ATLAS experiment [4] and adopted by other experiments including GLAST and HARP.

B. Separation Between Data and Algorithms

Even though we have decided to use object oriented technology we wanted to profit from all the accumulated experience in doing software in High Energy Physics experiments. In fact one of GAUDI's main design criteria is to distinguish *data objects*, whose basic function is to carry data, from *algorithm objects* which have an algorithmic role. While data objects will essentially provide manipulation of internal data members, algorithms will, in general, process data objects of some type and produce new data objects of a different type.

C. Algorithms

Algorithms are the essence of the data processing applications and where the physics and subdetectors code is encapsulated. Due to the fact that algorithms implement a standard set of generic interfaces they can be called without knowing what they really do. The *application manager* knows which algorithms to instantiate and when to call them. The algorithms execution is scheduled explicitly by configuring the application manager or by the execution of the *Data On Demand* service. Since complex algorithms can be implemented by using a set of simpler ones, a more elaborate control sequence can be established in the applications to support filtering and branches. These are combined with multiple output streams to provide event filtering and selections. The different LHCb data processing applications are customized by choosing the appropriate set of algorithms to execute.

D. Separation of Transient and Persistent Data

An important design choice has been to distinguish between a *transient* from a *persistent* representation of the data objects, for all categories of data. Algorithms see only data objects in the transient representation and, as a consequence, are shielded from the technology chosen to store the persistent data objects. In fact, so far, we have changed from ZEBRA [5] (for legacy data) to ROOT/IO [6] and more recently to POOL [7] without the physics code encapsulated in the algorithms being affected. The two representations can be optimized following different criteria (e.g., execution versus I/O performance) and different technologies can be accessed (e.g., for the different data types).

E. Transient Data Stores

The data flow between algorithms proceeds via the transient store. This not only shields them from the persistent technology but minimizes the coupling between independent algorithms, allowing their development in a fairly autonomous way.

We have distinguished between three categories of data. *Event data* are obtained from particle collisions and their successive processing, *detector data* describe the detecting apparatus (geometry, calibration, etc.), and *statistical data* result

from processing a set of events (histograms, n-tuples). They are not only conceptually different types of data, their access pattern, and their "lifetime" during a "job" is also different; hence we have organized them in corresponding separate transient data stores. The *Transient Event Store* contains the event data that are valid only for the time it takes to process one event. The *Transient Detector Store* contains the data that describe the various aspects of the behavior of the detector (e.g., alignment) during a period of data taking corresponding to the processing of many events. The *Transient Histogram Store* contains statistical data which typically have a lifetime corresponding to the data processed in a complete job. Although the stores behave slightly differently, i.e., the clearing of the store is handled at different frequencies in the three cases, their implementation is based on a common transient store component given the many things they have in common.

We have already mentioned that the data flow between algorithms proceeds via the transient store. In addition the transient store acts as an intermediate buffer for any type of data conversion to a different type of data representation, in particular the conversion to persistent or graphical objects. Zero or more persistent or graphical representations of the data can correspond to one transient representation.

The data within the transient store is organized in a "tree-like" structure, similar to a Unix file system, allowing data items that are logically related (for example, produced in the same processing stage) to be grouped together at runtime. Each node in the tree is the *owner* of everything below it and will propagate its deletion to all items in its branches. To map object oriented data models onto a tree structure, object associations have been implemented using symbolic links in which ownership of the referenced items is left to the node holding them.

F. Services

This category of components offers the services common to most of the applications. They are generally sizable components set up by the framework at the beginning of a job and used by the algorithms as often as needed. This approach allows the algorithm developers to avoid writing the routine software tasks that are typically needed in a physics data application.

Some examples of services can be seen in Fig. 1. More details on GAUDI and LHCb services will be given in Section III.

G. Tools

Tools are lightweight objects whose purpose is to help other components perform their algorithmic work. Sometimes an encapsulated piece of code can be executed with different frequency (only for some events or many times per event); it can be necessary to execute it on each data object separately or the data objects on which to perform the algorithmic operation could be local to the component. Finally different components may wish to share the same algorithmic operation *as is* or configure it slightly differently (e.g., different event selection algorithms will want to combine reconstructed particles to make vertices).

To provide this kind of functionality we have introduced a category of processing objects that encapsulate these "light" algorithms and called them *Tools*.

III. Core Services

The GAUDI framework is decomposed into a number of independent subframeworks to provide the routine software tasks typically needed in an application.

The basic kernel of the framework together with a set of utility services constitutes the *General Framework Services*. The *Job Options Service*, to configure the applications at runtime, the *Message Service*, the *Random Number Generator Service*, the *Event Data Service*, and the *Histogram Service* are some examples of what belongs to this category.

Other subframeworks provide specialized functionality. Many of these services use *third-party* components. This allows to profit from existing software and helps in minimizing development and maintenance efforts.

A technology-neutral *Object Persistency* mechanism has been developed and interfaced with the framework, given the fact that a single persistency technology may not be optimal in all cases. The persistency mechanism has been designed such that the best adapted technology can be used for each category of data. *Data Management and Bookkeeping* provides event tag collections and mass storage interfaces. The new LCG POOL framework is based on a similar architecture allowing the client code to be technology free. Last year POOL has replaced the LHCb ROOT/IO based persistency solution previously in place. This will allow us to benefit from the additional functionality provided by POOL such as file catalogues and event collections.

The *Data Dictionary Service* provides a high-level modeling language to define the event object model, independent of the language used in the current implementation of the processing applications (i.e., C++) [8]. The description language chosen has been XML, which provides a very strict syntax in addition to being very flexible. A GAUDI parser package (GAUDI Object Description) automatically produces the C++ header files. This approach ensures adherence to coding conventions, consistent sets of member functions, standard ways of cross-referencing objects, documentation's lines in the format required by the code documentation tool (Doxygen). The service also provides runtime introspection information for object persistency and interactive analysis making use of the LCG object dictionary provided by the SEAL project [9].

Event Model Support Classes like containers and reference classes are also provided to support the implementation of the LHCb Event model.

The *Detector Description* framework allows detector related information to be available to the physics applications providing a generic description of the structure of the geometry. The aim has been to have a unique description of the detector for all applications (e.g., simulation and reconstruction). The physical and logical description of the LHCb detector plus subdetector specific data resides in a Detector Description Database (DDDB) that provides the persistent storage of the detector data. Various versions of the DDDB following the evolution of the LHCb detector design have been produced.

Definition and implementation of interactive services, graphical interfaces, and scripting tools are provided in *User Interaction* services.

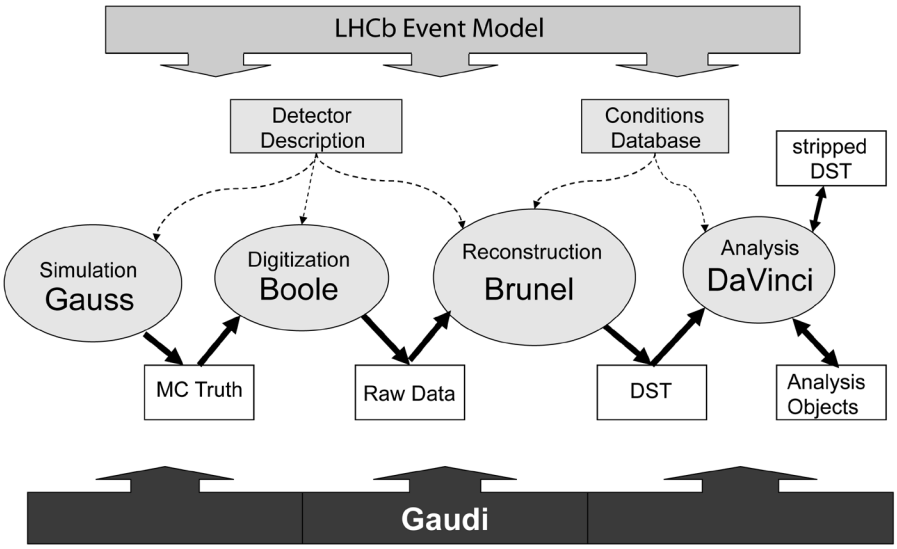

Fig. 2. The LHCb data processing applications and data flow.

Finally, specialized frameworks for simulation, analysis tools (not the tools themselves), and data visualization have been put in place.

A. The LHCb Event Model

The Event model is the description of the event data items that are exchanged between algorithms in any application. In order to obtain a coherent set of objects, a stable event model has been designed and implemented for all processing phases.

Guidelines for common approaches to the Event data model have been put in place and utility classes provided. Direct access to contained objects is done via a unique identifying key, the container to which objects belong allowing also sequential access. Explicit relationships between classes in the data model occur only between classes adjacent in the processing sequence. There is a clear separation between reconstructed data and their Monte Carlo truth origin, with no explicit links between the two; their relationship is preserved in special *linker/relations* objects.

The Event Data model classes are described using the high-level description language mentioned before.

The current LHCb event model contains the following sets of objects: Monte Carlo truth objects (generation and decay vertices, particles, hits in sensitive detectors), digitized information from sensitive detectors, raw data blocks, reconstruction objects (tracker clusters, energy clusters, tracks, etc.), particle identification objects, physics objects (particles, vertices, etc.). Particular emphasis has been put on all classes that have the likelihood of being made persistent.

The classes have been thoroughly reviewed before being implemented. Changes to the existing Event model in light of the experience with its use follow the same review procedure.

IV. Data Processing Applications

Typical phases of Particle Physics data processing have been encapsulated in the various LHCb applications. Each application is a producer and/or consumer of data for the other stages as shown in Fig. 2. The applications are all based on the GAUDI framework; they share and communicate via the LHCb Event model and make use of the LHCb unique Detector Description. This not only ensures consistency between the applications but allows algorithms to migrate from one application to another as necessary. The subdivision between the different applications has been driven by their different scopes (simulation and reconstruction) and convenience (simulation of the events and detector response) as well as CPU consumption and repetitiveness of the tasks performed (reconstruction and analysis).

A. Gauss, the Simulation Applications

Gauss [10] mimics what will happen in the spectrometer to allow understanding of the experimental conditions and performance. It integrates two independent phases that can be run together or separately. Normally they are run as a single job. Both phases make use of libraries and toolkits available in the Physics community.

The first phase consists of the event generation of proton–proton collisions and the decaying of the B mesons in channels of interest for the LHCb physics program. It is interfaced to Pythia [11] for the event production and to a specialized decay package, EvtGen [12]. It also handles the simulation of the running conditions (e.g., event pileup and primary vertex smearing due to the special luminosity the experiment will run at). Other event generator engines can be interfaced in this phase if required. The particles produced are stored in the HepMC [13] generic format that can be made persistent if this phase is run in *stand-alone* mode.

The second phase consists of the tracking of the particles produced in the proton–proton interactions in the LHCb detector. The simulation of the physics processes the particles undergo when traveling through the experimental setup is delegated to the Geant4 toolkit [14]. Geant4 interacts with Gauss using a set of interfaces and converters encapsulated in a GAUDI specialized framework (GiGa [15]) which allows the conversion of the LHCb detector geometry into the Geant4 geometry. It also converts the output of the first phase of Gauss to the Geant4 input format. The output of Geant4 in the form of hits produced in the sensitive detectors as well as the Monte Carlo truth history is

then converted back into the LHCb event model. The behavior of the Geant4 simulation engine in terms of detectors to simulate, physics models to use, details of the Monte Carlo truth to be provided, is controlled at runtime via job options configuration.

Gauss has replaced the previous FORTRAN-based simulation application at the end of 2003. After validating it both with test beam data and by comparison with its predecessor, it is now used for massive data production (data challenge).

Improvements to the simulation both in terms of the application itself, additional details and new features (e.g., machine background) are foreseen and are being continuously implemented.

B. Boole, the Digitization Application

The simulation of the detector responses and their digitization in order to produce data in the same format as the experiment electronics and DAQ system is provided by Boole. Although logically this process is part of the simulation, it is very convenient to integrate it in a separate application. Boole is also responsible for including in the data the effect of the adjacent beam crossings in the sensitive detectors. Simulation of imperfections in the detectors' response is continuously improved with the evolving knowledge acquired from test beam data.

Boole has been put in production as a separate application in 2003 and is part of the applications used in data challenges.

C. Brunel, the Reconstruction Application

Brunel integrates complete pattern recognition as well as subdetector and combined reconstruction. It produces as output files containing all reconstructed items such as calorimeter and trackers clusters, charged tracks, as well as information on particle identification from the RICH, calorimeter and muon subsystems (DST).

Brunel can process identically the results of the Boole digitization and data directly from the data acquisition system (DAQ) and as such is independent from simulation.

In addition to the input event data, Brunel needs to access the detector description and the conditions database (including alignment and calibration). Work is in progress, in collaboration with LCG, to deploy a dedicated framework for the conditions database.

Brunel is the oldest of the LHCb object oriented applications having been operational and of production quality since August 2002; however continuous changes to the pattern recognition strategy for performance improvements and additional functionality are in progress.

D. DaVinci, the Analysis Framework

The analysis framework supports selection of events and analysis proceeding from the further processing of the DST data. It provides tools of general utilities for the manipulation (e.g., vertexing) and analysis of the physics event objects that are described in term of "particles" and "vertices." In addition, tools to allow the evaluation of the physics performance of the code are provided to enable study and comparison with the Monte Carlo truth information.

The output of DaVinci can be purely statistical or event data. Analysis object data files containing physics objects can be written to allow further processing. The output of DaVinci can also be a reduced DST, where only events satisfying certain conditions are written. Selection algorithms can be integrated into a complete DaVinci selection job dynamically configured: a production version of this job is in place to further process the data produced and is used for event selection production in the data challenge.

V. Software Triggers

The structure of the GAUDI architecture, with algorithms never communicating directly with permanent data storage, makes it also well suited for online applications where data comes from the DAQ. Only the Input Service to the Transient Store, the job control, and the monitoring components need to be specialized to interface with the DAQ and with the Experiment Control System, while other components can be used identically as in offline applications.

The software trigger algorithms are currently being developed in the DaVinci application to evaluate their performance, while the components to be specialized for the online environment are developed in parallel.

VI. Visualization

The graphical display of detector geometry and event data objects is provided by a dedicated application called Panoramix.

It is based on a set of visualization services and converters providing the graphical representation of the LHCb setup as well as of the data. The event data can be read from files or produced on the fly. An interactive user interface allows the user to choose what to display and how it is visualized. The visualization services are based on the OnX[1] package for interactivity and Open Inventor[2] for the graphics. Python is used as the scripting language to control the GUI and to provide the necessary functionality by wrapping LHCb C++ code. Predefined views have been implemented and are available in the GUI as well as the normal zoom and rotation facilities.

Since Panoramix is based on the GAUDI framework, the LHCb detector description and event model, it can work with any of the data processing applications described before allowing not only 3D graphical rendering for geometry verification but also providing aid in the development and understanding of the physics algorithms.

VII. Conclusion

Software is a fundamental aspect of an experiment and has a strong impact on its success. The software development approach chosen is architecture driven and has proven to be flexible and resilient to change. Applications for the various processing stages have been implemented and, although in different stages of development, are all used for massive production of data (more than 210 million events) for studies to be carried out

[1]http://openscientist.lal.in2p3.fr

[2]http://oss.sgi.com/projects/inventor/

in the experiment. The applications are also routinely used by all LHCb collaborators in their everyday work.

The GAUDI framework is expected to adapt to the changes that will occur in the third-party software used (e.g., POOL) as well as replace dedicated components with common libraries with similar functionality (e.g., adopting the component model provided by LCG), thus minimizing the software maintenance.

Additional services will be provided as the physicist requirements evolve, making use as much as possible of external software (e.g., linear algebra and minimization via LCG provided software).

The need to ease the development and debugging of the physics software as well as possibility of performing interactive analysis is driving the choice of using Python[3] as scripting language. Generic Python bindings for the GAUDI framework provide access to the objects in the C++ world combining the full functionality of the existing framework with the flexibility of the Python language. Further development with Python for easier interactive control of the application and access to additional LHCb services is foreseen.

Adopting a common framework and services has allowed maximum reuse of the software and helped in minimizing duplication of code. The necessary evolution of some of the services provided is expected to have minimal impact on the applications due to the interface model adopted.

REFERENCES

[1] S. Amato, "LHCb technical proposal," CERN-LHCC-98-004, 1998.

[2] R. Antunes Nobrega, "LHCb reoptimized detector—Design and performance," CERN-LHCC-2003-030, 2003.

[3] G. Barrand, I. Belyaev, P. Binko, M. Cattaneo, R. Chytracek, G. Corti, M. Frank, G. Gracia, J. Harvey, E. van Herwijnen, P. Maley, P. Mato, S. Probst, and F. Ranjard, "GAUDI—A software architecture and framework for building HEP data processing applications," *Comp. Phys. Comm. 140*, 2001.

[4] W. Amstrong, "ATLAS technical proposal," CERN-LHCC-94-43, 1994.

[5] R. Brun, "ZEBRA Reference Manual," *Program Library Q100, CERN*, 1991.

[6] R. Brun and F. Rademakers, "ROOT—An object oriented analysis framework," *Nucl. Instrum. Methods Phys. Res A*, vol. 389, pp. 81–, 1997.

[7] R. Chytracek, D. Dülmann, M. Frank, M. Girone, G. Govi, J. T. Moscicki, I. Papadopoulos, H. Scmuecker, K. Karr, D. Malon, A. Vaniachine, W. Tanenbaum, Z. Xie, T. Barrass, and C. Cioffi, "The LCG POOL development and production experience," presented at the IEEE-NSS, Rome, Italy, Oct. 2004.

[8] M. Cattaneo, G. Corti, M. Frank, P. Mato, S. Roiser, and S. Miksch, "Event data definition in LHCb," presented at the CHEP03, San Diego, CA, Mar. 2003.

[9] J. Generowicz, P. Mato, L. Moneta, S. Roiser, M. Marino, and L. Tuura, "SEAL: Common core libraries and services for LHC applications," presented at the CHEP03, San Diego, CA, Mar. 2003.

[10] I. Belyeav, Ph. Charpentier, S. Easo, P. Mato, J. Palacios, W. Pokorski, F. Ranjard, and J. van Tilburg, "Simulation application for the LHCb experiment," presented at the CHEP03, San Diego, CA, Mar. 2003.

[11] T. Sjöstrand, P. Edén, C. Friberg, L. Lönnblad, G. Miu, S. Mrenna, and E. Norrbin, "High-energy physics event generation with Pythia 6.1," *Comp. Phys. Comm.*, vol. 135, pp. 238–, 2001.

[12] D. Lange, "The EvtGen particle decay simulation package," *Nucl. Instrum. Methods A*, vol. 462, pp. 152–, 2001.

[13] M. Dobbs and J. B. Hansen, "The HepMC C++ Monte Carlo event record for high energy physics," *Comp. Phys. Comm.*, vol. 134, pp. 41–, 2001.

[14] S. Agostinelli, "GEANT4—A simulation toolkit," *Nucl. Instrum. Methods A*, 506, pp. 250–, 2003.

[15] I. Belyaev, M. Frank, G. Gracia, P. Mato, and F. Ranjard, "Integration of Geant4 with the GAUDI framework," presented at the CHEP2001, Beijing, China, Sep. 2001.

[3]http://www.python.org/