THE LHC++ ENVIRONMENT

Bernardino Ferrero Merlino
CERN IT Division - CH 1211 Geneva 23 Switzerland

Abstract
The LHC++ project is an ongoing effort to provide an Object-Oriented software environment for future HEP experiments. It is based on standards-conforming solutions, together with HEP-specific extensions and components. Data persistence is provided by the Objectivity/DB Object Database (ODBMS), while the IRIS Explorer Visualization system is the foundation for the Interactive Analysis environment. To complement the standard package, a set of C++ class libraries for histogram management, ntuple-like analysis (based on Objectivity/DB) and for presentation graphics (based on Open Inventor) have been developed.

1. INTRODUCTION

Over a period of many years, CERN, in conjunction with other laboratories, built up a large collection of routines and programs oriented towards the needs of a physics research laboratory. This software – almost entirely written in Fortran, is referred to collectively as the CERN Program Library or CERNLIB [1]. For many years, it was assumed that CERNLIB would simply be migrated from Fortran 77 to Fortran 90. However, in the early ’90s an important change took place, namely the adoption of object-oriented techniques and programming languages such as C++ and – more recently – Java. As a result of these changes, the need for the “C++-equivalent of CERNLIB” arose. The LHC++ project was initiated in 1995 to address these issues. Given the falling manpower envelope of the laboratory, it was clear that there would be insufficient resources to develop and support everything in-house and so alternatives, such as collaborative development and the use of commercial components, were investigated.

The current LHC++ [2] strategy relies on both commercial and HEP-specific components. It’s noteworthy that the LHC++ environment is built using a ‘layered’ approach, where all basic functionality are implemented as standalone C++ class libraries that are then integrated using a more sophisticated Modular Visualization System (MVS). A sketch of the LHC++ components is given in Table 1 below:

Description	Components
Data Analysis	IRIS Explorer - HEPExplorer
Custom graphics	MasterSuite - HEPInventor
Basic graphics	OpenInventor - OpenGL
HEP math	HEPFitting – GEMINI - CLHEP
Basic math	NAG C library
Histograms	HTL
Database	HepODBMS
Persistency	Objectivity/DB
C++	Standard Libraries (STL)
HEP specific	CLHEP

Table 1 - LHC++ Components

2. LHC++ COMMERCIAL COMPONENTS

Many factors contributed to the choice of the commercial components of LHC++. These included the functionality of the individual packages, their adherence to standards – either *de-facto* or *de-jure* – their interoperability, their market share (including other HEP laboratories) and of course cost! Several of the suppliers chosen already had a long-established relationship with CERN from previous software packages and the systems themselves were "interrelated". This is important as it not only guarantees their interoperability but simplifies the issues related to ensuring consistent releases across multiple platforms – these issues having been already addressed by the vendors concerned.

2.1 Objectivity/DB ODBMS

In order to study solutions for storing and handling the multi-PB data samples expected with LHC, the RD45 Project [3] was established in 1995. The proposed solution should also be able to cope with other persistent objects, such as histograms, calibration and monitoring data, and so forth. It was found that the best candidate for handling this problem is an Object Database Management Group (ODMG) [4] compliant object database used together with a mass storage system, based upon the IEEE reference model for mass storage systems [5]. After considering a few alternatives, the presently favored solution is built upon Objectivity/DB [6] and HPSS (High Performance Storage System) [7].

2.2 IRIS Explorer

IRIS Explorer [8] is a toolkit for visualization of scientific data, which can be manipulated via visual programming tools. Users define their analysis application by connecting building blocks, called modules, into a so-called map (see Figure 1 below). Modules act like filters: they read one or more streams of input data and produce one of more streams of output data. The behavior of modules is controlled (interactively) by a set of parameters. IRIS Explorer comes with a rather complete set of modules for performing basic data transformations and it is straightforward to create new modules. IRIS Explorer is built on top of recognized graphics standard such as OpenGL [9] and Open Inventor [10], thus making possible to integrate third party packages based on the same standards, e.g. GEANT-4 [11].

2.3 OpenGL

OpenGL is an industry standard for graphics. It is vendor-neutral and multi-platform, and is optimized for building environments for developing 2D and 3D visual applications. Several vendors already offer a hardware implementation of the standard, thus ensuring that rendering speed will be optimal.

2.4 Open Inventor

Open Inventor is an object-oriented 3D toolkit built on top of OpenGL, providing a comprehensive solution to interactive graphics programming. Its programming model is based on a 3D scene database optimized to ease building graphics applications. It includes a large set of objects, such as cubes, polygons, text, materials, cameras, lights, track-balls, handle boxes, 3D viewers, editors and defines a standard file format (IV) for 3D data interchange files, that is the basis for the Virtual Reality Modeling Language (VRML) [12] standard.

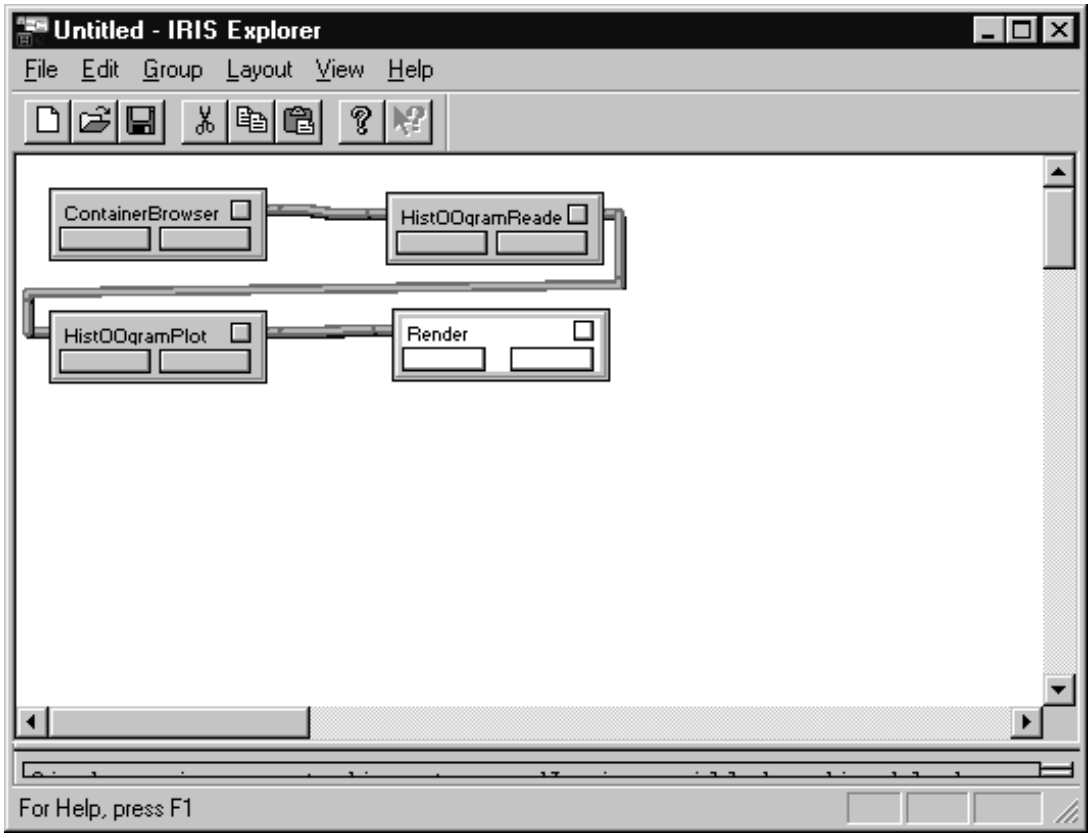

Figure 1 – example of a "Map" in IRIS Explorer

2.5 MasterSuite

MasterSuite [13] is a C++ toolkit for data visualization, containing class libraries with extension nodes to Open Inventor. These extensions cover both 2D (drawing, charting, etc.) and 3D (drawing, legends, etc.). In addition, it provides a set of classes to develop viewers for scientific data for output on screen as well as in vector-PostScript format.

2.6 NAG C Library

The NAG C Library [14] is a collection of about 400 user-callable mathematical and statistical functions. The library includes facilities in the area of minimization, ordinary differential equations, Fourier transform, linear algebra, zeros of polynomial, statistics, time series etc. The library uses double precision throughout to ensure maximum accuracy of results. The correctness of each library function is evaluated and verified by specially written test programs performed on each of the machine ranges for which the library is available.

3. LHC++ HEP-SPECIFIC COMPONENTS

Although the commercial components on which LHC++ is built offer a solid foundation, they do not – in general – provide the complete functionality that is required in the HEP community. To cater for such needs, small extensions – typically some 2-3K lines of code – are provided.

3.1 HepODBMS

HepODBMS [15] is a set of class libraries built on top of the ODMG C++ interface. Their purpose is to provide a higher level interface than is specified by the ODMG, to simplify the porting of existing applications and provide a minimum level of support for transient-persistent switching. Furthermore, these libraries help to insulate applications against changes between releases from a given vendor and between the products of different vendors. The HepODBMS libraries provide classes to deal with session management, clustering hints, tag and event collections.

3.2 The Histogram Template Library (HTL)

The Histogram Template Library (HTL) [16] is a C++ class library that provides powerful histogram functionality. As the name suggests, it exploits the template facility of C++ and is designed to be compact, extensible, modular and fast. As such it only deals with histograms (summary data representing the frequency of values) and not with the whole set of values. Furthermore, although simple file-based I/O and "line printer" output are supported, it is not coupled to more advanced I/O and visualization techniques. In the context of LHC++, such capabilities are provided by other components that fully interoperate with HTL.

HTL itself offers the basic features of HBOOK [17] as well as a number of useful extensions, with an object-oriented (O-O) approach. These features include the following:

- booking and filling of 1D, 2D and profile histograms;
- computation of statistics such as the mean or r.m.s of a histogram;
- support for operations between histograms;
- Browsing of and access to characteristics of individual histograms.

3.3 HEPInventor

HEPInventor [18] proposes an easily understandable and user-friendly way to present data in physics programs. It is implemented as a graphical class library built on top of MasterSuite to provide an interface between HTL and its presentation graphics.

3.4 HEPExplorer

HEPExplorer [19] is a set of HEP-specific IRIS Explorer modules, which help a physicist set up an environment to analyze experimental data, produce histograms, fit models and prepare data presentation plots. IRIS Explorer Maps that implement simple analysis-related tasks, such as visualize and fit a histogram, produce histograms out of tag data (see section 4), etc. are part of the package as well.

3.5 Gemini/HEPFitting

Gemini [20] is a class library providing basic minimization/fitting capabilities. The library integrates under the same interface both MINUIT [21] and NAG minimizers, although classes are provided to access features that are unique to one minimization engine (such as NAG support for linear and non-linear constraints).

HEPFitting [22] is a utility library to fit either HTL or vectors of data, with a handy interface to specify complex fit functions assembling gaussian, polynomial or exponential terms, as well as user defined functions.

3.6 CLHEP

A set of HEP-specific foundation and utility classes such as random generators, physics vectors, geometry and linear algebra is packaged in the CLHEP class library [23]. CLHEP is structured in a set of packages independent of any external package (interdependencies within CLHEP are allowed under certain conditions).

4. ANALYSIS SCENARIO

The analysis scenario can be split in two parts. The first part concerns populating the database with reconstructed event data and is usually done in a C++ program, typically running in batch jobs. The second part implies using an interactive tool, such as the IRIS Explorer framework, to actually produce summary data, usually as histograms, out of the event data. Histograms can then be manipulated, fitted using an appropriate tool and eventually printed in a PostScript file to embed in a paper or a slide presentation.

4.1 The 'batch' part

The main task of this part is populating the Objectivity data store with event information coming, very likely, from a former reconstruction phase. Most new HEP experiments assume that it will be possible to make both raw data and reconstructed data available on-line thanks to the integration between Objectivity/DB and HPSS. Each experiment will have its own data model and physicists should be able to navigate through it. This is a major problem for a general-purpose Interactive Analysis environment, since, unlike the Ntuple case, a common and pre-defined data model, shared amongst all experiments, is no longer imposed.

Since all data needed for analysis is supposed to be on-line, the role of the Ntuple replacement could be quite different. While reasonably small personal data collections will still exist, the main concern will probably be how to index large event stores to speed up the analysis.

The RD45 Project suggested one approach to deal with both problems. The idea it to speed up queries by defining for each event a Tag, i.e., a small set of its most important physics attributes plus an association with the event where the Tag data come from. A collection of tag objects is saved together in a Tag Database, something intermediate between an Event Directory and an Ntuple. Since they are globally defined for the whole experiment, concrete tags can be optimized so that they offer a very efficient way to make initial cuts on attributes, thus achieving a high degree of selectivity. On top of that, at any moment users can cross the association to the event to retrieve any other details about the full event, which are not contained in the Tag.

In general the experiment or group will make the selection of key attributes characterizing events, so that concrete tags are mostly defined for experiment-wide or workgroup-wide data sets. However, individual physicists have the possibility to define their own simpler data collection by using the Generic Tag mechanism. This second lightweight procedure allows users to define a tag on the fly, without creating a persistent class. Compared to the concrete tag, there is, of course, a small performance penalty, but this is most of the time balanced by an increased flexibility, since at any time new fields can be added to the tag and the association to the complete event data remains available.

The set of individual tags is called an Explorable Collection, i.e., a collection of objects implementing an interface for access from IRIS Explorer.

4.2 An example: creating a Tag collection out of existing events

The Event we want to create the Tag from is composed by two kinds of information:

- Tracking information, represented by a variable size array of tracks
- Calorimeter information, represented by a variable length array of clusters

```
/// persistent Tracker class
class Tracker : public ooObj {
public:
ooVArrayT<Track> tracks;
private:
};
/// persistent Calo class
class Calo : public ooObj {
public:
ooVArrayT<Cluster> clusters;
};
/// persistent Event class
class Event : public ooObj {
private:
  int evtNo;
public:
  d_Ref<Tracker> tracker;
  d_Ref<Calo>    calo;
};
```

So, for each event, we will have a collection of tracks and a collection of clusters, plus a unique event identifier.

The classes implementing a single track or cluster will contain information related to the particle traversing the two sub-detectors:

```
// Basic track: persistent by embedding
class Track {
public:
double getPhi()   { return phi;}
  double getTheta() { return theta;}
  double getPt()    { return pt;}
private:
  double phi;
  double theta;
  double pt;
};
// Basic cluster: persistent by embedding
```

```
class Cluster {
public:
 double getPhi()    { return phi; }
  double getTheta()  { return theta; }
  double getEnergy() { return energy; }
private:
  double phi;
  double theta;
  double energy;
};
```

The tag we want to create will contain the pT and phi attribute of the tracks having maximum and minimum pT, plus the event unique identifier. Hence the Tag description will be something like:

```
HepExplorableGenericTags highPt; // create a tag collection
// define fields all fields that belong to genTag
   TagAttribute<long>  eventNo (highPt,"eventNo");
   /* track with highest pT*/
   TagAttribute<double> ptPlus (highPt,"ptPlus");
   TagAttribute<double> phiPlus(highPt,"phiPlus");
   /* track with lowest pT*/
   TagAttribute<double> ptMinus (highPt,"ptMinus");
   TagAttribute<double> phiMinus(highPt,"phiMinus");
```

It’s now possible to scan the events, identify the tracks with minimum/maximum pT and replicate their pT and phi attributes in the Tag:

```
ooItr(Event) eventItr;
eventItr.scan(container("Events"));
while( eventItr.next())
  {
   HepRef(Tracker) aTracker = eventItr->tracker;
   int maxTrack = 0,  minTrack = 0;
   for (int track=0; track < aTracker->getNoOfTracks(); track++) {
    if (aTracker->tracks[track].getPt()
       > aTracker->tracks[maxTrack].getPt() )
      maxTrack = track;
    if (aTracker->tracks[track].getPt()
       < aTracker->tracks[minTrack].getPt() )
      minTrack = track;
   }
highPt.newTag();      // create a new tag (all fields have default values)
 eventNo  = eventItr->getEventNo();
 ptPlus   = aTracker->tracks[maxTrack].getPt();
 phiPlus  = aTracker->tracks[maxTrack].getPhi();
 ptMinus  = aTracker->tracks[minTrack].getPt();
 phiMinus = aTracker->tracks[minTrack].getPhi();
}
```

It is noteworthy that the Tag’s attribute are managed exactly as standard C++ variables: the overloaded assignment operator will take care of putting the values in the Tag that will be stored in the database.

4.3 The interactive part

Interactive Analysis in IRIS Explorer is implemented by a set of HEPExplorer modules. Generally speaking, the current set of modules allows users to extract data from an Objectivity/DB data store and put them in one or more HTL histogram(s). In particular the user can select an Explorable Collection, define a set of cuts over the collection as a C++ expression, define the input streams for the HTL histogram(s) to produce and automatically generate and compile C++ code that implements the cuts and fill the histograms.

Apart from accessing the data in the tag, users can invoke C++ functions that implement, e.g., common physics or access the experiment specific event object (by traversing the association between a tag and its related event). User-defined functions can be used whenever a C++ expression is allowed. This means, for example, that reconstruction C++ code can be used in the analysis module (and the other way round).

Since there's no interpreter involved, the analysis code can use any C++ feature supported by the local compiler (templates, STL, exceptions, etc.)

An alternative to code generation/dynamic compilation is the use of a restricted C++ syntax to specify the cuts. Such restricted syntax is then interpreted to filter the data that will fill the histogram. An example of such approach is the TagViewer module (see Figure 2 below):

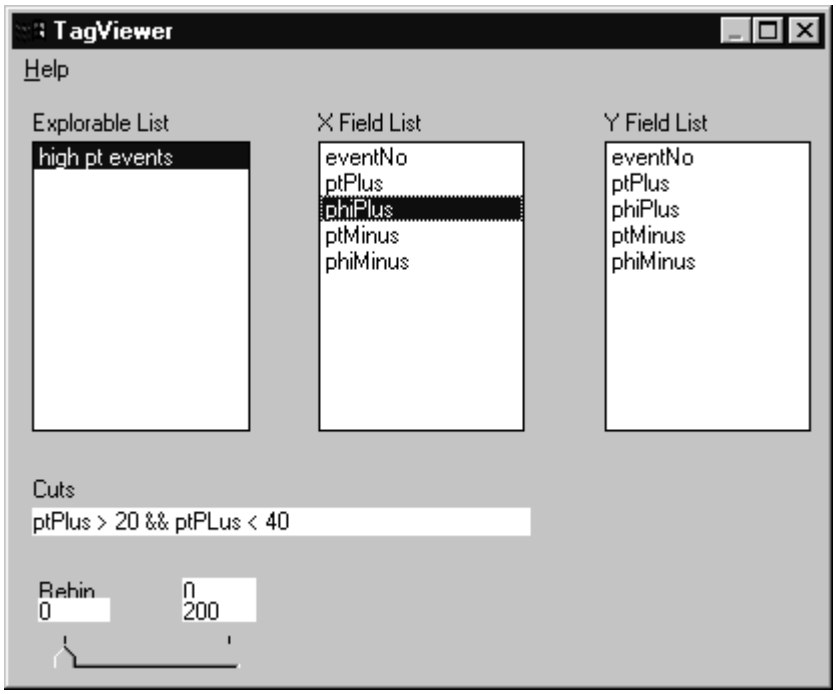

Figure 2 – The TagViewer Module

The cuts are expressed as a simple C++ expression involving only Tag variables, relational and logical operators.

5. CONCLUSIONS

The HEPExplorer package is a successful effort to integrate IRIS Explorer and Objectivity/DB so that high energy physicists can 'exercise' the analysis chain, from event to paper, on data stored in an object oriented database.

We believe IRIS Explorer is a good environment for data analysis and visualization: its compliance with graphics standard, its simple development environment, its robustness and modularity being certainly the main good points.

The layered approach has proved to be an effective way to cope with change.Since the first release of LHC++ (July 1998) we have already changed the basic C++ libraries (from Rogue Wave's Tools.h++ to STL) and the whole histogram package (from HistOOgrams to HTL) without major impact on any other part of the package.

REFERENCES

[1] CERN Program Library (CERNLIB): see http://wwwinfo.cern.ch/asd/index.html.

[2] LHC++: see http://wwwinfo.cern.ch/asd/lhc++/index.html

[3] RD45 - A Persistent Object Manager for HEP, see http://wwwinfo.cern.ch/asd/rd45/index.html.

[4] The Object Database Management Group (ODMG): see http://www.odmg.org/.

[5] The IEEE Storage System Standards Working Group: see http://www.ssswg.org/.

[6] Objectivity/DB: see http://www.objectivity.com.

[7] The High Performance Storage System: see http://www.sdsc.edu/hpss/.

[8] IRIS Explorer User Guide, ISBN 1-85206-110-3, 1995.

[9] OpenGL Reference Manual, ISBN 0-201-63276-4, Addison Wesley, 1992.

[10] OpenInventor Reference Manual, ISBN 0-201-62491-5, Addison Wesley, 1994.

[11] An Object-Oriented Detector Simulation Toolkit (GEANT-4): see http://wwwcn.cern.ch/pl/geant/geant4.html.

[12] Virtual Reality Markup Language (VRML): see http://vrml.wired.com/.

[13] 3DMasterSuite: see http://www.tgs.com/.

[14] NAG C library: see http://www.nag.co.uk/.

[15] HEPODBMS: see http://wwwinfo.cern.ch/asd/lhc++/HepODBMS/user-guide/ho.html.

[16] HTL: see http://wwwinfo.cern.ch/asd/lhc++/htlguide/htl.html.

[17] HBOOK: see http://wwwinfo.cern.ch/asdoc/hbook_html3/hboomain.html.

[18] HEPInventor: see http://home.cern.ch/~couet/HEPInventor_doc/.

[19] HEPExplorer: see http://wwwinfo.cern.ch/asd/lhc++/HepExplorer/index.html.

[20] GEMINI: see http://wwwinfo.cern.ch/asd/lhc++/Gemini/.

[21] MINUIT - Function Minimization and Error Analysis Package, CERN Program Library Long Writeup D506: http://wwwinfo.cern.ch/asdoc/minuit_html3/minmain.html.

[22] HEPFitting: see http://wwwinfo.cern.ch/asd/lhc++/HepFitting/.

[23] CLHEP - A Class Library for HEP, see http://wwwinfo.cern.ch/asd/lhc++/clhep/index.html

[24] PAW - the Physics Analysis Workshop - CERN Program Library Long Writeup, Q121