OpenScientist. Status of the project.

G. Barrand, LAL, Orsay, France*

Abstract

We present the OpenScientist project [1] ; its basic driving rules, the technological choices and the modules that permits to visualize various source of HEP data. We present OpenPAW, an emulation of the CERN PAW program. At the end, we compare to ROOT, examine the position of LCG and terminate by interogating the role of CERN in software for physic.

WHAT IS OPENSCIENTIST ?

OpenScientist is first of all an INTEGRATION of packages working together to do scientific visualization and data analysis. It is NOT one million of lines of home made code and then not a reinvention of everything. In particular, it cannot be considered as another ”//The ROOT of EVERYTHING” [2].

DRIVING RULES

This integration is driven by the following rules :

- Follow the ”software least action principle”. It consists to reach a maximum of functionalities by minimizing the number of home made code and number of packages in general.
- For GUI and graphic, use what desktop providers offer (to have full speed).
- For the rest, use a maximum of dedicated open source software.
- Target the local user desktop machines. It means to be fluent on Linux(es), MacOSX and Windows.
- Have a flexible architecture to be able to EVOLVE. This is a crucial point for HEP knowing that doing an experiment is a matter of decades.

C++

We stick to C++ because event model managers (data frameworks) are in C++ and that C and its derivative (C++, ObjectiveC) are still the native languages of desktop providers for GUI and graphic. Any other language would induce penalties in accessing data or local resources.

HAVE A FLEXIBLE ARCHITECTURE

This goal is achieved by having ”hub” packages to do the coarse graining couplings and by using pure abstract interfaces to minimize these couplings.

VISUALIZATION PRODUCTS

The rendering layer is done by using OpenGL which is now supported by all desktop providers. As a scene manager we have adopted OpenInventor by using the Coin3d implementation of System In Motion (Norway) [3]. All the graphic, 2D and 3D, is done with OpenInventor and OpenGL. In particular the plotting of histogram is done also with OpenInventor. This unification of all the graphic permits to build interesting views like having a piece of detector or part of an event in a plot, or immersing the plot of an histogram in the display of an event.

GUI, A PAIN

Finding a good strategy for the GUI is a more painfull situation since the ”OpenGL miracle” never happened for the GUI. No standard API emerged from desktop providers. This is due to the fact that desktop software is still the horse battle of major actors : Microsoft with Windows, Apple with Cocoa (in fact NextStep), GNU with GNOME/gtk and KDE with Qt. If aiming the common desktops around, the life of a developer become quickly a nightmare when having to find a solution. WE NEED SOME STANDARD API coming from desktop providers (some common XML ?) .

GUI, THREE STRATEGIES

We may think to develop our own GUI toolkit (like ROOT) but this will bring us back to the age of stone. We may think to choose a cross platform toolkit, but at long this appears to be a frustrating solution ; someone fond of [Windows,Cocoa,KDE,GNOME] wants to use [Windows,Cocoa,KDE,GNOME] on his machine.

Then we are going to try to use what desktop providers offers, that is to say use Windows on a Windows, NextStep on a Mac, Gtk on a Linux/GNOME, Qt on a Linux/KDE and Motif on other UNIXes. Must be point out here that Motif is still the fastest solution on a Linux... This way of doing will offer obviously the best performances on local machines. Fine, but then how to proceed ?

GUI IN XML, THE ONX PACKAGE

We have decided to describe the GUI in XML : widget hierarchy and callbacks. For example, a piece of GUI looks like :

* barrand@lal.in2p3.fr

```
  <widget class="PushButton">
    <label>My hello button</label>
    <activate exec="Python">
      from import OnX *;ui.echo("hello")
    </activate>
  </widget>
```

From an XML description of the GUI, the OnX package creates it by using native toolkits.

OnX could be seen as some omni-interface-builder (from an XML description of the GUI). Drivers exist now for Xt/Motif, Windows, gtk, Qt and NextStep. OnX could be seen also as a factory for these GUI toolkits. Note that a huge difference with desktop provider interface builders (glade, Qt-Builder) is that there is no code generation from the XML (the XML being some internal file format for these GUI builders). With OnX, the XML is part of the application and visible from users. This permits obviously a strong customization of the GUI without recompiling.

ONX AND INVENTOR

The strong point of OnX is that it does the integration of Inventor viewers. This had been done by using the SoXt, SoWin, SoQt, SoGtk, SC21 packages of System In Motion. To create a viewing area, someone simply put in its XML :

```
  <widget class="PageViewer"/>
```

The PageViewer is a sofisticated widget able to handle multiple viewing regions (movable and resizable with the mouse). Being an Inventor examiner viewer, it benefits of the famous Inventor thumbwheels to manipulate a 3D scene.

SCRIPTED CALLBACKS

In the XML, someone put also the widget callbacks. In the upper example, a Python script is executed when clicking on the ”My hello button” button.

A lot of scripting system had been declared to OnX. The C system function, permitted to spawn shell commands :

```
    <activate exec="sys">ls</activate>
```

The dynamic loader :

```
    <activate exec="C++">
      OnX ui_echo hello
    </activate>
    <activate exec="C++">
      MyDLL my_cbk arg1 arg2
    </activate>
```

In the second activate, the extern C entry point my_cbk is searched in the MyDLL dynamicly loadable ”callback” library and executed with the following arguments (here arg1 and arg2). The my_cbk code must be of the form :

```
extern "C" {
  void my_cbk(IUI& aUI
  ,const std::vector<std::string>& aArgs) {
    // User code
    aUI.echo("my_cbk executed !!!");
  }
}
```

the argument aUI permits to access all GUI components and also the OnX ”ISession” that permits to access the various OnX managers (ScriptManager, TypeManager, LibraryManager, MemoryManager in Lab, etc..) but also managers declared by users. OnX comes with a default OnX callback library containing various predefined callbacks like the upper ”ui_echo”.

Python had been declared to OnX. The wrapping of C++ code(for example abstract interfaces IUI, ISession, etc..) is done by using SWIG. KUIP had been declared too (it comes with OpenScientist). Someone can have :

```
  <activate exec="kuip">
    h/plot the_famous_10
  </activate>
```

CINT had been declared too. By crossing fingers, someone can execute :

```
  <activate exec="CINT">
    {AIDA::IAnalysisFactory* aida = ...}
  </activate>
```

THE LAB PACKAGE. DATA ANALYSIS

OnX, by integrating GUI, graphic and scripting is the ”hub” package for interactivity. The hub package for data analysis is the Lab package.

AIDA

Lab is AIDA-3.2.1 compliant [4]. Lab can produce XML files containing histograms and tuples and then exchange these files with other AIDA implementations like the jas one written in java. Lab can export also files at the ROOT format with the histogram streamed like the TH of ROOT. These files are readable by jas and ROOT.

HCL histograms

The LAL HCL package is used for histograms. It is a light and dedicated package that appears to be 20 percent faster than the ROOT TH classes. The difference of speed comes to the fact that HCL uses internally std::vector which appears to be faster in optimized mode than the TArray of ROOT. The HCL::Histogram class is a truly multidimensional histogram class over STL. In particular in HCL, a 1D object does not have forever dummies TAxis fYaxis, fZaxis fields irrelevant for a 1D object that someone can find in ROOT TH1 class ! The HCL::Histogram class does not inherit graphical classes (like the THs that inherit TAtts). The visualization of histogram is done in the Lab package by using he HEPVis/SoPlotter nodekit.

Rio for storage

The LAL Rio is a light and clean package for doing IO at the ROOT format. (What is a file at the ROOT format ?). With it OpenScientist is now ROOT free. Note that in file produced with Rio, the Lab::Histograms are not streamed like ROOT TH histograms. This is due to the fact that a HCL (or Lab or AIDA) histogram is more rich that a TH ; then streaming it like TH would mean a loss of information. For data exchange with ROOT (and jas) someone have to use the ”export” format, so that histograms are streamed like TH.

Fitting

OpenScientist uses now the C++/Minuit done at CERN.

Plotting

The plotting is done with Inventor, by using the SoPlotter nodekit developed at LAL by the author and deposited in the HEPVis CVS repository of Fermilab. This plotter supports now most of the common plotting cases, in particular contour plotting and, obviously, 3D lego and surfaces. The vector PostScript production is done by using gl2ps.

OPENPAW

The opaw program is an OnX application to emulate PAW.

```
OS> opaw  [my.kumac]
OS> opaw -gui [my.kumac]
```

OpenPAW could be seen as a PAW interactive front end to AIDA and AIDA could be seen as the C++ API to OpenPAW.

Commands

Due to the fact that KUIP had been taken (for long) from the old CERNLIB and that the Pawcdf.cdf, describing PAW commands, had been taken from old CERNLIB too ; someone has the SAME syntax as PAW. With OpenScientist-13.0, the original pawex1.kumac up to pawex24.kumac had beeen emulated with, in general, better performances on most aspects. Obviously not all commands (and options) are yet recovered but things are under way.

COMIS

COMIS had been replaced by ”on the fly” compilation and loading. It works with FORTRAN but also with C and C++ without having the burden of interpreters.

Vectors and SIGMA

Vectors and SIGMA commands are here too. Vectors had been reimplemented by using Lib::Vector, a multidimensional vector class over STL. The SIGMA command had been done by using the Lib::Processor expression evaluator (done with lex and yacc). Note that when evaluating something like V1*V2, the OpenPAW SIGMA command loops internaly directly within std::vector, then ensuring a maximum of speed.

OTHER MODULES

Geant4

The G4Lab provides visualization for Geant4 data : geometry, trajectories and physic tables. Geometry and tables can be browsed in a GUI tree widget and leaf element be clicked and displayed in the document area of the GUI. Geometry are visualized by using Inventor and a physic table is histogrammed and displayed by using the HEPVis/SoPlotter.

Geant3

G3Lab provides visualization for Geant3 geometries. Like Geant4 geometries, they can be browsed in a GUI tree widget and displayed with Inventor by clicking leaf elements. It is used at LAL on Windows and very appreciated.

RELEASES

Experience shows that the use of dedicated open source packages clearly minimize the coding and improve the quality. But building the integration in order to come with a consistent set of packages easily installable on three heterogeneous platforms is clearly not free of work. Up to now OpenScientist is released twice a year, in June (summer release) and December (winter release). Major changes being done in general in the summer release.

WHAT IS REALLY UPSETING IN ROOT ?

Too much flaws

We have already mentionned the TH1 having forever dummies TAxis fYaxis and fZaxis (see AIDA for a better design). The IO buffer accesses are not protected on overflow (Bytes.h, tobuf methods). Streamers do not have a return status in case of problem. On a corrupted file, the crash (or exception carpet hiding) is unavoidable. Some simple protection (done in Rio) would permit to treat the problem in a clean way (stop the streaming, giveup this file, warn the user, etc...). The TArray is definitely much slower that an std::vector. The list of problem is long but probably the most awkward is to find a Draw method on the basic introspection system ! None of the object oriented language or software that the author had in hand have that (C++, java, ObjectiveC, Csharp, Qt, Inventor, NextStep, Python, etc...). Is the TClass::Draw() a breakthrough in software coming from HEP ? The author does not believe that.

Follow the maximum action principle

In ROOT, a maximum of things are home made. In front of all good quality dedicated things available for free now ; why, in 2004, do we have to burden forever badly secretly designed and inefficient classes for storage, histogramming, GUI, graphic scene manager, string, math, etc... ?

What is the place of home institute engineers in ROOT?

It is difficult to accept the borrowing of CERN name [5] for publicity whilst engineers of institutes that participate to the CERN program had not been consulted on the overall design and had seen their request for a major revision of ROOT for the LHC put aside. The only task left for them now for the next twenty years is the day to day debugging of an ever growing home made production. The situation is the same than being in front of a private company product. Is CERN a private software company ?

The ”LCG uses ROOT” decision

CERN / LCG did not order a unification of engineering researches done in various experiments around data frameworks. This was definitely an historical mistake. It was only a question to compel five to six men to put aside their personal ambition and to work for six to twelve months together on converging their results and products ; converging being not the same that choosing one by eliminating others. In particular, this was the last hope to compel a major revision of ROOT in order to federate engineers (and not only physicists) around one appealing common basement for the LHC. CERN HAD NOT BEEN ABLE TO DO THAT. Instead came the ”LCG uses ROOT” decision. Already this decision induces an astouding amount of complication at the level of the code. There are two dictionaries ; the LCGDict, a dedicated introspection system (fine) and the one embedded somewhere in CINT and wrapped in the ROOT TClass(::Draw). The needed POOL system clearly suffers of that. It would be more simple to have, behind POOL, a dedicated IO package (at the ROOT format ?) and use the LCGDict only (then avoiding the full burden of a C++ interpreter here). There are also the presence of two plugin managers that are incompatibles (Seal/PluginManager, TPluginManager). We do not even mention the problem of the various build systems. The author had to port all that on MacOSX (plus Gaudi and the whole LHCb software on top of it) : this had been months of astounding pain. All this is a sociological and engineering shame : the LHC does not deserve that.

What is CERN ?

For the author [7], CERN had been created, first of all, to federate scientific and engineering forces of a set of countries to run (big) experiments to do physics. Part of accelerators and detectors, after agreements between members, are build in home institutes or private companies and assembled at CERN, where this place offers the infrastructure and organization to assemble and operate all that. SOFTWARE is clearly an engineering deliverable and, what have to be provided and maintained is clearly out of scope of one man alone. Then software must follow the same paradigm than machine and detectors. It means, in particular, that this system can’t work by putting engineers and technicians in pure concurrency from begin to end. At some point a convergence on technologies must be done in order to synchronize ALL PARTIES ; and this because the gifted manpower is in fact spare and that everybody is needed. How much men today have views on the internal of ROOT and SEAL and Gaudi and POOL and Geant4 : it could be counted with fingers of one hand. A set of fermionic developers must become at some moment a bosonic group. ”LCG uses ROOT” (another way to say ”take it and shut up”) shows that some people, part of the program, are out of engineering control. By violating the idea of federation, it signs the death of CERN as an engineering federating place for doing scientific software. By giving engineering immunity to some, it opens, once more, the way to another era of poor engineering and overall mediocrity concerning software questions.

CONCLUSIONS

OpenScientist integration works and is sufficently used to continue. The opposition with ROOT is huge because the strategies adopted are at two extremes ; the ”make all at home” for one and ”integrate dedicated open source code done elsewhere” for the other. The author is deeply convinced that the second option is more adapted to the today world and definitely more adapted for a lab like CERN created to federate engineering forces of countries to run experiments.

REFERENCES

[1] http://www.lal.in2p3.fr/OpenScientist.

[2] TROOT.cxx

[3] http://www.coin3d.org

[4] http://aida.freehep.org

[5] Linux Journal of July 1998 : ”ROOT an OO data analysis framework. A data analysis tool developed and used by CERN”

[6] But it seems so also for Robert Aymar : see ”point c” in CERN Courier March 2004. ”CERN’s role on the European stage”