Rio

G. Barrand, LAL, Orsay, France*

Abstract

Rio (for ROOT IO) is a clean and light C++ rewriting of the file IO system of ROOT [1]. We shall present our strong motivations of doing this tedious work along with the main choices done in the Rio implementation. We shall say why Rio is more reliable that ROOT. We shall present the effort done around Gaudi in 2002 to read LHCb events with Rio. We shall present our views about the POOL LCG proposed solution of a storage system based on ROOT. To finish we shall explain why CERN is going to miss an essential target in software : an appealing open source object oriented storage system for HEP.

MOTIVATIONS

This work had been motivated by the strong conviction that an IO package is not a drawing package. We do not believe that a dedicated IO package on which someone can find a Draw() method on its introspection system is good engineering. Helas, the minimum that we can extract from ROOT to do the IO comes with the TClass having a Draw() method on it. On none of the object oriented language or package that the author had in hand (C++, java, Csharp, NextStep, Qt, Inventor) someone can find such thing. Is the TClass::Draw() a breakthrough of humankind coming from HEP ? Or is it an indication showing that the engineering of ROOT and then of its IO system is globally not a good one.

POTENTIAL CLIENTS

Rio is intended for people that looks for an open source dedicated file IO system ROOT compliant, but do not want to enter in the whole ROOT system. Potential clients are people that develop their own framework (event model manager) and seek for a well defined dedicated IO package which is not another framework.

HISTORY

Previous attempts (Rio-v1r*, Rio-v2*, RioGrande-v3*) were a repacking of the "ROOT core" library. But it appeared that the ROOT core is in fact the core of a framework and then more than an IO package. Usage of these previous Rio versions within the OpenScientist Lab [2] package showed that it was still necessary to bring around 200 000 lines of code to store and retrieve an histogram in a file in a non reliable way. This involves the code of the IO machinery but also the code of the CINT interpreter used mainly at this point to have an automatic generation of the streamers of a couple of classes. It involves also to bring on board some codes to handle networking, drawing, GUI, etc... that are in fact irrelevant to the problem of storing data in a file. Some non-discussions with the ROOT conceptors clearly showed that these people will not do any attempt in order to have a more modular repacking of the ROOT core. The idea of doing Rio was born in November 1998 after the CHEP'98 conference of Chicago.

DICTIONARY

The author was interested in studying the question of the relationship of the CINT interpreter with the IO machinery. Is CINT really needed ? Isn't it possible to have the IO machinery repacked in order to be able to use an abstract dictionary so that someone can provide the dictionary info by hand, or in an automatic way with other interpreters or languages that have introspection ? The mastering of the dictionary production would permit in particular to be able to reuse the dictionary machinery of some languages like Python and java to store objects of these languages with a minimum of code between these languages and the IO machinery. Be able to write a dictionary by hand can permit some software not interpretable by CINT (Geant4, heavily templated code, etc...) to have access to a storage system. Some framework, like Gaudi, uses the SEAL LCGDict dedicated reflexion package. It could be fine to be able to use these dictionaries in direct connection to the IO package

PURE ABSTRACT INTERFACES

The author was interested also to study the usage of pure abstract interfaces ; a technique that permits to decouple domains in a nice way at the level of the code. The critical points to study were the coupling of the dictionaries to the IO system and of the data streamers to the IO system. We remember that a pure abstract interface permits to establish a relationship at compilation time but not at link time. Imagine ; having NOT to link your DLLs with ROOT libCore, libCINT, libTree, etc...! Due to the strong resistance of the ROOT team to not use this nice technique, the author wanted to know if this resistance was justified technically. The answer is no. With Rio-v3r0 the goal had been achieved ; the IO machinery sees only some pure abstract classes like Rio::IDictionary, IClass, IObject and the data

* barrand@lal.in2p3.fr

streamer sees the Rio::IBuffer. What is astounding (compared to ROOT) is that these classes have really few methods. In particular the Rio::IObject and Rio::IClass have NO draw method. Note that speed, being highly dominated by the system read/write, the overhead of using pure abstract interfaces is neglectible in this problem.

STL

It was clear also that some code in the ROOT core was here due to a wish of not using existing libraries like STL and to a tendency of the authors to rewrite, reinvente or T-repack most of things. Was it not possible to have something lighter by simply using STL ? The answer is obviously yes.

CLASSES

Rio is light because it does in 15 klines of code what ROOT does with 200 klines. It is clean because, among other things, it has 85 classes and 9 interfaces organized in a clean inheritance tree dealing only with the problem of IO. This has to be compared to the needed 228 classes of ROOT having for the moment zero interfaces.

FILE ORGANIZATION

The logic of the IO itself had been respected as much as possible (at least up to the understanding that the author has of the IO logic of ROOT). In particular the streamers of the basic classes like TFile, TKey, TDirectory, TTree had been respected so that a file produced by Rio may be understood by ROOT itself. For example a file containing a TTree filled with basic data types like ints and floats is readable by ROOT.

FORWARD COMPATIBILTY WITH ROOT

What happens with files produced with the brand new last release of ROOT ? Is Rio able to read them ? This is not guarantee for the moment. One painfull point is that basic data organizer streamers (TDirectory, TTree, TBranch) are oftenly touched in ROOT and some time for reasons not related to storage. For example, the TBranch had received recently the inheritance of TAttFile which is irrelevant to the problem of the IO (the ROOT team thinks probably that the IO of a TBranch is more efficient if drawn with hatches). The TBranch streamer being changed then we can consider that the file format had changed. Someone can reply by saying that the streamer infos of a TBranch is stored in the file and, by exploiting that, someone would be forward compatible with new ROOT version. Right, and we are going to do that in Rio, but anyway it would be welcome to have some guarantees that ROOT file format be not drastically changed at each release for fancy reasons. Who can enforce that ?

CODING DRIVING RULES

In Rio there is no ”g” logic that is to say no global pointers and singletons. In ROOT, the fact that classes may see other classes through global pointers clearly breaks the encapsulation. Any classes in ROOT can establish relationships to any other classes in a non traceable way. We even don’t speak of relationship established by using the CINT interpreter (string relationship). In Rio, the relationships are established only by using inheritance, encapsulation and methods arguments (as explained in all good book about OO).

Having no singleton, someone can instantiate two Rio::Files by having the guarantee that there is no hidden relationships between them. It means that someone can have true multithreading on multiple files in good confidence.

We do not provide, in Rio itself, an automatic streamer production by using some interpreter. This should come as third party packages. That’s right that for the moment we have no one for Rio.

We do not have in Rio networking, visualization, GUI, etc... All these are other problems. One consequence is that the relationship to the operating system specific things is minimal. It is concentrated in the Core/File.cxx file and concerns mainly the C functions : open, close, read, write, lseek. The configuration and installation is then straightforward.

We use STL, then we do not reinvent string, list, vector, etc...

We avoid pointers as much as possible and then use (const) references. We use the ”Rio” namespace, then we have Rio::Xxx instead of TXxx. We namespace the libraries, that is to say we have libRioCore, libRioTree instead of libCore, libTree, etc... that may clash with other products. We have IXxx to name an interface (IClass, IObject). We use basic data types (int, short, double, float) ; we do not reinvent all the basic data types.

And once more we have no Rio::IClass::draw() or Rio::IObject::draw() methods. An IO package is NOT a drawing package.

We avoid also static objects and then static object constructors. This permits to build safe DLLs on all platforms. We try to be ANSI C++. The code had been tested with five ANSI C++ compilers (g++, VisualC++, DEC/cxx, Sun/CC, KCC). We avoid technicalities and eXtreme C++. Experience shows that it is not intelligible for others and most of the time breaks the portability.

If you find that all the upper points are common sense, then contact someone of the ROOT team and try to discuss these points with him.

RELIABILITY

In ROOT, the IO buffer accesses are not protected on overflow, especially around strings (see in Bytes.h the 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 protections would permit to treat the problem in a clean way (stop the streaming, give up this file, warn the user, etc...). In Rio this had been done and experience shows that Rio is much more difficult to crash than ROOT.

RESULTS

The results had been VERY encouraging. Now the OpenScientist Lab package is able to store/retrieve histograms and tuples with around 11 000 lines of code only. Files containing TTree with simple data types (int, float) are not only readable by ROOT itself but also with the java implementation of the ROOT IO reader done by T.Johnson at SLAC [3].

In September 2002, we have been able to read LHCb data produced at the ROOT format during summer 2002. These data had been produced within the Gaudi framework, by using the GaudiRootDb service and with ROOT-3-01-06 behind. The data have been read by using a modified version of GaudiRootDb (GaudiRioDb) in order to use Rio-v3r0. In LHCb, it appears that this way is now no more practicable since this experiment had decided to use LCG/POOL (see below). Anyway, someone else using the Gaudi framework has now a very light IO package at hand to handle file storage.

AT CERN

The existence of Rio (and the fact that it was able to read LHCb data in 9 / 2002) had been notified to main actors of storage at CERN and LHCb in 9 / 2002. At that time the reached solution to read LHCb data was simple and clean. It summed up to one Gaudi service (GaudiRioDb package) over Rio. Put all together it amounts up to around 20 klines of code only. The reactions had been unanimous and non scientific : we do not want to ear about it !

Now CERN (and LHCb) goes in the direction of some kind of fermionic mixture with ROOT to do the IO and some upper layer (POOL over SEAL) to handle collections of files. All this dealing with two dictionaries that will obviously never been merged, three incompatible plugin systems (Gaudi, SEAL and ROOT ones), three build systems, etc...

Right now the amount of code to read an LHCb event is probably around one million of home made code, covering CINT, ROOT, SEAL, POOL, Gaudi and not counting "external" packages like : boost, gccxml, pcre, uuid, zlib, mysql++, rx. A pain.

The author invites the reader to have a look at the code (or, let us be kind, only the class diagrams) and try to understand how a piece of data is stored in a file with all that. The author had to do the port on MacOSX of all the uppers in order to run the LHCb Panoramix visualization system ; it had been months of intrinsic intellectual pain. Will it be needed to wait 2030 that blocking people be retired in order to have another chance to put things on track and have something appealing concerning HEP storage software ? Are we going to be saved, before that, by a third party dedicated open source product ?

For the 50th birthday of CERN, the author wishes to the lab that had been created to federate engineering forces to do high energy physics, and claims to have the "E" of European in its name, a very good luck with the storage of data of the LHC experiments

FUTURE

Rio is used in the OpenScientist Lab package and OpenPAW program. The developments will follow the needs that will come around these softwares. Are already requested the chaining of tuples and the storing of more AIDA data types. It is clear that Rio covers IO in one file only. The logic would be to continue by handling operations on collection of files and then be lead to a data base software. For the moment, only the name of this package had been found : RioGrande ! Any volunteer to help doing it with the same coding spirit that the Rio package ?

DOWNLOAD

Rio comes with the OpenScientist distribution. It can be reconstructed with CMT or configure on UNIXes and with CMT or .NET on Windows.

CREDITS

Despite the differences of opinion on everything, the author thanks anyway Rene Brun and Fons Rademakers that designed, around 1994, the ROOT IO format which appears to be anyway efficient (on speed and size of files) to store HEP data.

CONCLUSIONS

Rio is used in OpenScientist (and OpenPAW) to store data. We definitely would like that this work be an impulse to make aware physicists and computer scientists that a rethinking of ROOT is possible on a much more clean and simple basement that what exists today.

REFERENCES

[1] http://root.cern.ch

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

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