OpenPAW

G. Barrand, LAL, Orsay, France*

Abstract

OpenPAW is a CERN/PAW [1] emulation done with the OpenScientist integration principles and tools [2] [3]. OpenPAW is for people that definitively do not want to quit the PAW command prompt, but seek anyway an implementation based over more modern technologies.

SAME SYNTAX THAN PAW

The C part of the KUIP package had been extracted (for long) from the old CERNLIBS. It means that specific KUIP commands like exec, shell, edit, etc... are here. The pawcdf.cdf of the CERNLIBS, describing the PAW commands, had been used so that someone has the guarantee to have the same syntax than PAW. The ”only” thing that remained to do was to reimplemente the ”callback” C functions pahist, panntu, etc.. that does the concrete work behing commands. This had been done by using AIDA [4] and the OpenScientist implementation of AIDA. Then OpenPAW could be seen as an interactive front end to AIDA but AIDA could be seen also as the programming API to OpenPAW. Obviously not all commands and options are yet implemented, but things are underway. See the web page for a list of commands that received an implementation. Today the CERN/PAW tutorial examples pawex1.kumac up to pawex24.kumac are already emulated with quite same rendering than PAW. (In fact there is the exception of pawex19 dealing with ”tuple mask” that is not yet here along that ”set NDVX”).

TECHNOLOGIES

The technologies are the one of the OpenScientist integration, then we shall refer to the OpenScientist CHEP’04 paper for a more detailed description of choices. To sumup ; the rendering layer is OpenGL, the scene manager is Open Inventor, the GUI builder is OnX. The GUI is described in XML and the OnX package is the factory that permits to build concrete GUIs by using the toolkit of native desktop providers (Windows, Cocoa, Motif, gtk and Qt). The default scripting used for OnX and Lab packages is the system ”dld/C++” one, but Python can easily be plugged in. Obviously, for the purpose of OpenPAW, KUIP had been declared to OnX so that someone can put in its GUI XML something like :

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

* barrand@lal.in2p3.fr

PLOTTING

The plotting is done by using also OpenInventor (yes, Inventor can be used for doing 2D !). The HEPVis SoAxis and SoPlotter OpenInventor nodekits permit to handle a plotting scene with axes. The XY, Lego, function, contour plotting are here. The quality, especially 3D, is in general better than CERN/PAW. For the 3D someone can easily manipulates the scene by using the famous Inventor thumbwheels. For paper output, OpenPAW can produce pixmap PostScript of an Inventor viewer but also vector PostScript by using gl2ps. With gl2ps, files are in general big but they are WYSIWYG which is a strong point, especially with lego or surfaces. Note that doing remote X11 graphic is not recommended : USE YOUR LAPTOP.

STARTUP

After installation (see web pages), someone type :

```
OS> opaw
```

to start with a PAW-like GUI (one prompt and one viewer).

```
OS> opaw -gui
```

to start with a more complete and compact GUI ”a la PowerPoint”.

WHAT ABOUT COMIS ?

COMIS had been replaced by ”on the fly compilation and loading” of FORTRAN, C, C++. It means that the full syntax of these languages are available at full speed. The first invocation of a source file has the overhead of compilation and DLL building, but further executions are fast. For the moment someone cannot reload a DLL without quiting. This would permit to modify the code, recompile and reload on the fly without quiting. The reloading of DLLs passes by the mastering of the unloading of them and of the related code ; an operation that has to be studied with care. This is in the joblist...

SIGMA

Vector handling is done with the Lib::Vector template class. Lib::Vector is a multidimensional vector template

done by using std::vector (then fast). The SIGMA command is done with the little Lib::Processor algebra interpreter. It is done with lex and yacc and aware of the Lib::Vector ; in particular something like V1*V2 loops directly within std::vectors : fast.

FILE FORMATS

The situation here is a little bit confuse and needs some explanation in order to understand what could be done today by OpenPAW and what to use in batch programs to produce files understood by OpenPAW.

Batch, working with AIDA

For batch programs we recommand to use the AIDA API to create histograms and tuples, fill them and put them in files at various formats. The file formats depend of the AIDA implementation you use. The OpenScientist AIDA implementation is the Lab package. In writing, Lab handles today two file formats : Lab/ROOT (done by using Rio) and AIDA/XML. OpenPAW can obviously read these files but also other formats (see below).

Lab/ROOT

"Lab/ROOT" must be understood in the sense that the organization of the file is the ROOT one (TKey, TDirectory, TTree), but the objects put in the files are Lab ones ! In particular, by default, the Lab::Histograms (AIDA complients) are not streamed like THs objects. It had not been possible to do that because AIDA histograms are more rich than the THs. The differences are a direct consequence of the unability to discuss the nature of histograms with the ROOT team in a neutral place. Note that Lab can also stream out its histograms like THs but this has to be seen as an "export" mode (to jas [5] or ROOT [6]). Writing and reading back Lab::Histograms by using THs streaming will induce a lost of information. Note that OpenPAW can read ROOT3 files containing THs and simple TTrees containing real numbers. This probably does not work anymore on ROOT4 files since the basic data organizer streamers are oftenly touched in ROOT (for example to inherit them with some TAttFill irrelevant to the problem of IO (here the ROOT team thinks probably that the IO of a TBranch is more efficient if drawn with hatches...)).

Lab/ROOT and tuple writing

The Lab tuples are today written like TTree with branches of floats. These could be read by ROOT. In principle OpenPAW can read simple TTree exported by ROOT, at the condition that streamers be not changed at each new ROOT version for fancy reasons.

ROOT file, we need a standard here

The upper sections show that we need to fix the internal organization of a ROOT file but also that we need to fix the streaming of some basic objects like histograms and simple tuples. Who can order that ?

AIDA/XML files

In a batch, someone can produce AIDA/XML files with various AIDA implementations : OpenScientist, FreeHEP (done in java) and LCG/PI [7]. In principle OpenPAW can read these files.

Zebra files

Helas we do not have a Zebra rewritten in C or C++ ! (Which should have been done for long at CERN). But someone can reconstruct the Zebra driver of the OpenScientist/Lab package by linking the old CERNLIBS. With that someone can read hbook files within OpenPAW. Note that HBOOK objects will be converted to in-memory Lab objects (including tuples !). In batch, the ex-ANAPHE or LCG/PI AIDA implementations can produce these files.

H/FILE

The h/file OpenPAW implementation can then open all these files :

```
opaw> h/file 2 my_file.root
opaw> h/plot 10
opaw> close 2
```

FITTING

h/fit and ve/fit received an implementation, but clearly not with all options. Behind, the AIDA fitting interfaces are used with, behind them, the LCG C++/Minuit dedicated fitting package.

PERFORMANCES

The targeted machines are local desktop (Linuxes, Macs, Windows). On these machines OpenPAW offers better performances than the CERN/PAW on most aspects (in particular graphic).

WHY OPEN ?

OpenPAW is open for various reasons. The first one is that it is "spiritualy open" in the sense that we try to follow the OpenScientist strategy of "integrating dedicated open source code done elsewhere". Another one is that the architecture and the code attempts to be very modular. This is done by using pure abstract interfaces to do the coupling between domains (GUI, scene manager, scripting, storage). We remember that a pure abstract interface permits to estabish a relationship between two classes without the need to link the "using class" library against a specific implementation of the "used class". The relatioship is done at compilation time and at run time only (through dynamic

loading of a DLL doing an implementation of the interface). Right now we can handle various GUIs, scripting (KUIP, Python), and storage (Rio, AIDA-XML) in a clean way. The packaging of OpenPAW is also open because someone can pickup most of the ”facilities” without having to embarque connection to irrelevant things (storage without the graphic, etc...). It is also open to user in the sense that someone can strongly customize a lot in a simple way without having to touch everything. And at last, but not the least, it is open because all the code is open source.

CONCLUSIONS

We had presented the first ”serious” release of OpenPAW. It contains sufficient material to work. It demonstrates also that we can offer a continuity to physicists at the same time that, behind the scene, software engineers have the freedom to move toward new technologies. It is clear that not all what was available in CERN/PAW is yet recovered but on numerous points someone has already much more.

REFERENCES

[1] http://wwwasd.web.cern.ch/wwwasd/paw

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

[3] CHEP’04 OpenScientist paper.

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

[5] http://jas.freehep.org

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

[7] http://lcgapp.cern.ch/project/pi