Data Analysis – a Physicist's Perspective

J. Swain *Northeastern University, Boston, MA 02115, USA*
L. Taylor *Northeastern University, Boston, MA 02115, USA*

Abstract

We describe the traditional HEP physics analysis environment in terms of the data types and the software tools used and, using a typical analysis task as an example, we identify the generic problems which are encountered. We then attempt to define requirements for a future analysis environment considering such issues as correctness, homogeneity, consistency, fault tolerance, ease of use, and efficiency. Finally, we discuss the hopes and plans for a future physics analysis environment based on an object oriented software paradigm.

Keywords: Physics data analysis; visualisation; object oriented paradigm.

1 Introduction

To function in the typical physics analysis environment used in HEP today requires an extensive familiarity with a wide range of tools and data types in a highly heterogeneous framework. We examine the situation and then attempt, with some unavoidable hubris, to suggest ways in which a future analysis environment might alleviate some of the problems described and also enhance productivity by using modern computing technologies. Since this subject has been discussed a great deal amongst the HEP computing community, we will emphasise the perspective of the physicist. The eventual goal of any physics analysis is to make measurements of fundamental physical quantities which can then be compared to theoretical models. Such quantities are rarely determined directly – in general raw event data must first be processed in a variety of ways and by a variety of different tools. Figure 1 shows schematically the flow of data, represented by arrows, between the traditional software modules used for HEP offline analysis, represented by rounded boxes. The box labelled

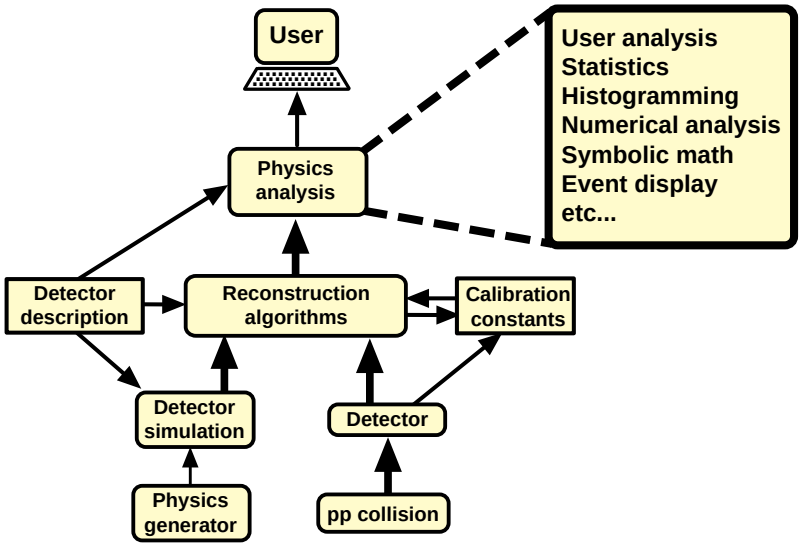

Figure 1: Schematic flow of data between the main software tasks of a typical physics analysis.

"Physics analysis" contains many elements which are not specific to a particular experiment, as well as dedicated tools such as event visualisation programs. In addition to this "final analysis" step, analysis also includes: detector calibration and alignment; data quality checking; processing and selection of subsamples of real and simulated data based on physics criteria; numerical and statistical analysis; and comparison to theoretical models.

There are many data samples required for physics analysis and an even larger number of data formats:

- event data, including raw event data (typically sequential access binary files), reconstructed event data (typically ZEBRA [1] or BOS [2] files), and micro DST data (typically ZEBRA, BOS, or HBOOK ntuple files [3]);
- test beam data (a variety of obscure and poorly documented formats);
- detector description (various CAD files for the full engineering description and ASCII titles file, ZEBRA file, or FORTRAN 77 code when used with GEANT [4]);
- slow control data (ASCII, ZEBRA, binary files and/or a database such as Oracle or HEPDB);
- calibration data and run parameters (stored as for slow control data);
- natural constants such as π, e, *etc.* and fundamental constants such as c, $\hbar$, *etc.* (typically defined in multiple places throughout the code with varying degrees of precision);
- experimentally determined physics quantities such as particle masses, lifetimes, branching fractions, *etc.* (FORTRAN 77 code, ASCII files, PDG [5] database – often inconsistent and with no simple update mechanism);
- theoretical formulae such as matrix elements, fragmentation and structure functions, *etc.* (FORTRAN 77 code, output from a symbolic math package, printed papers).

There are a multitude of software tools used in even a simple analysis. The HEP-specific tools include:

- event generator programs (JETSET/Pythia [6], KORALZ [7], ISAJET [8], *etc.*);
- detector simulation programs (usually, but not always, based on GEANT [4] which is also used in its interactive form to analyse and display the detector elements);
- event reconstruction programs (almost completely specific to the experiment);
- statistical analysis programs (usually based on MINUIT [9], these may be written by the user or use general packages such as PAW [10] or mn_fit [11]);
- numerical analysis tools (generally libraries of routines, *e.g.* NAGLIB [12]);
- generic analysis software (the predominant package in use is PAW);
- data presentation software (the predominant package in use is PAW);
- event display software (these are specific to each experiment).

Many of the tools are essentially single-purpose while others, notably PAW, are applicable to a variety of different tasks. In addition, there are many tools used in physics analysis which are often taken for granted, such as the basic operating system functions, mail, printing, web browsers, graphics viewers, text editors, document preparation tools, and symbolic math packages.

2 A Typical Physics Analysis Task

In this section we describe a typical specific analysis subtask and the problems which arise. The conclusions we draw, however, will be general in nature. The analysis consists of finding photons produced in the neighbourhood of a jet. The photons are identified in a crystal calorimeter but are difficult to identify due to the overlapping energy deposits of the hadrons in the jet, as shown in figure 2(left). Electromagnetic showers are characterised by a narrow shower compared to hadronic showers, as shown in figure 2(right). Therefore, in order to identify a photon, one searches for a narrow energy bump allowing for a small background of dispersed energy deposits from hadrons. The electromagnetic shape is measured in the test beam while the hadronic shape is taken from a shower library derived from a GEANT [4] simulation study.

The optimum value for the fraction of electromagnetic energy is obtained by maximising the likelihood for the data to agree with the linear superposition of the two shower shapes. The candidate shower from the data consists of 25 energies deposited in a 5×5 array of crystals. Similarly, the ideal electromagnetic and hadronic shower shapes are each described by 25 real numbers. Conceptually the task is straightforward although in practice there a number of problems, as we describe below.

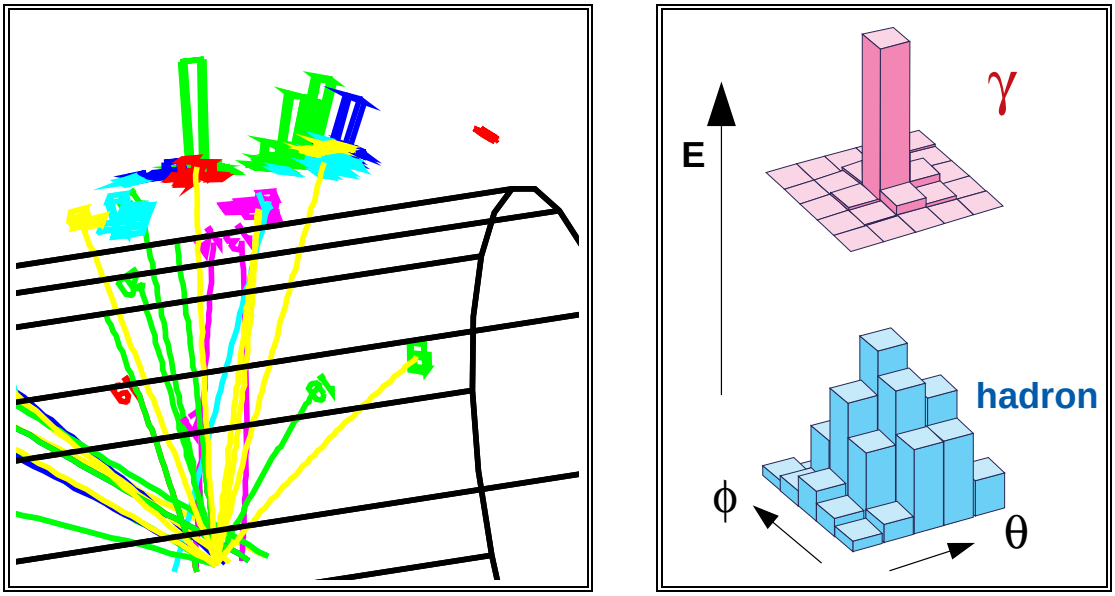

Figure 2: Actual shower in the L3 BGO calorimeter (left). The heights of the histogram-like bumps are proportional to the energy deposited. The figure on the right shows typical profiles for electromagnetic showers (upper) and hadronic showers (lower).

2.1 Problems encountered

In our simple example of shower shape fitting the following input files are required:

- the data events;
- the test beam data for the electromagnetic response function;
- a Monte Carlo file of simulated hadron responses;
- the geometrical parameters of the crystals;
- the run characteristics which define whether the calorimeter was functioning correctly;
- the calibration constants used to convert from ADC counts to energies, allowing for electronics pedestals and noise.

Thus, in order to perform one small aspect of a full analysis, making a one-dimensional fit involving 75 real numbers per shower, we need to read six files. This is probably inevitable, but the problem is that the format of the data is different in each case such that a different interface module is needed to read each file. To perform the MINUIT fits, `REAL` numbers then need to be converted to `DOUBLE PRECISION`. The results are converted back to `REAL` numbers and used to fill histograms which are written out in ZEBRA format for final plotting in PAW.

In addition to the multitude of different data formats, problems encountered include:

- nonuniform interfaces to the event display and reconstruction programs, PAW, MINUIT, the system shell, *etc.*;
- the low reliability and poor documentation of the programs used;
- the need to use the operating system to manage files and manipulate their formats;
- unnecessary repetition of work due to poor propagation and standardisation of work previously done by other physicists (*e.g.* detector alignments, interface routines *etc.*).

PAW was a well-motivated attempt to provide a unified physics analysis environment. Unfortunately the implementation suffered from many of the problems mentioned above which we elaborate, in the form of requirements,in the following section.

3 General Requirements for an Analysis Environment

In this section we attempt to define, from a physicist's point of view, a set of requirements for a coherent and easy to use physics analysis environment.

3.1 Correctness

The single most important requirement for the physicist is that results produced from analysis programs are correct. All too frequently this is not the case. A great deal of time is wasted by physicists trying to understand spurious results produced by faulty software or data (admittedly often their own). Of course in practice one must allow for intrinsic limitations of analysis programs, due for example to rounding errors and the pseudo-randomness of random number generators. These effects should be, and generally are, much smaller than the intrinsic error on the measurement.

3.2 Homogeneity

A significant fraction of a physicist's time is spent becoming familiar with the large variety of languages, data formats, interfaces, and associated tools, such that frequently programs are not used to their full extent if at all. It is desirable to offer an analysis environment which is as homogeneous as possible to minimise the learning time for each application. In particular, the data should be presented in as consistent a way as possible. The data themselves may be stored in a variety of formats but this should be hidden from the user. Use of a data description language (DDL) and a uniform command interpreter may greatly enhance the physicist's productivity.

3.3 Consistency

Closely tied to the issues of correctness and homogeneity is that of consistency which refers to the logical behaviour of commands and languages. For example, a language which is used to communicate instructions to a program must have a grammar in which every command parses in an *unambiguous* and *conventional* way.

3.4 Fault tolerance

Fault tolerance is the ability behave meaningfully in the presence of errors. Data used by a program should be checked for its validity since it may be incorrect or unavailable due to human error, error in code which generated it, or network or other hardware errors. Similarly, code should be written with checks to avoid such problems as overwriting, overflows and underflows, and memory leaks. The common practice of globally trapping such errors is extremely bad. The program should terminate gracefully, with meaningful diagnostic messages, rather than continuing with unpredictable results.

3.5 Ease of use

All too frequently physicists are required to learn unnecessarily obscure aspects of programming languages, data formats, or operating system functions . While this will always be inevitable for a few "experts" it is currently the norm rather than the exception. Commands and graphical user interfaces should be intuitive to use and have a style and syntax similar to other applications with which users are familiar. The documentation of the programs and data structures is of paramount importance, both for the developer and for the user. It should be an integral part of the software engineering design and implementation strategy and, in order to be useful, it should be up-to-date, accurate, and complete.

3.6 Software and Hardware Efficiency

The efficiency of code is nowadays not such an issue as it may once have been. At one time, physicists would optimise the code by hand to enhance the performance and tune the data structures to save disk space. Nowadays the lower cost of CPU's, memory, and disks, compared to the cost of physicists' time, makes it unprofitable to indulge excessively in these activities. Moreover, physicists are generally less competent than commercial compilers or profiling tools for analysing and optimizing code performance, and often less efficient and reliable than software or hardware compression algorithms for reducing the sizes of data sets. In general, the largest sources of inefficiency result from the need to repeatedly re-run programs due to errors in the code or data structures.

4 Development of a Future Analysis Environment

The large HEP collaborations of the future, such as ATLAS and CMS at the LHC, each consist of several thousand scientists dispersed around the globe. The analysis software typically consists of $\sim 10^{5-6}$ lines of code, which is developed and used over a period of $\mathcal{O}$(decades) on a multitude of hardware platforms. There is also a considerable amount of other software: DAQ, slow control, information systems, collaboration support tools such as personnel databases. The traditional HEP software model, whose problems we described above, is inadequate in satisfying the needs of these future experiments.

The HEP community has (almost) universally accepted that future software developments should use an Object Oriented (OO) paradigm [13, for example], using C++ at least for the moment. All relevant data (events, slow control data, calibrations, *etc.*) will be stored in an object database [14]. The OO strategy has many advantages; in particular the code and data structures are highly modular, the interfaces are clearly defined, and the software is (in principle at least) better engineered and documented. This approach will not only enhance the quality, usability, and maintainability of the code but it will also present a much more flexible environment for physics analysis. For example, in the CMS computing model, the serial data processing of the past (to form DST's, mini-DST's, *etc.*) will disappear at a logical level such that physicists may analyse raw data together with highly processed reconstructed objects. There are many generic HEP OO projects which are already well advanced, including LHC++ [15], GEANT4 [16, 17], and RD45 [18].

The use of commodity software components, for example for data presentation, will be encouraged wherever they fulfil the requirements. This will result in large reductions in manpower costs by removing the time consuming in-house development time, such that physicists can concentrate on those tasks which are specific to high energy physics. New paradigms of distributed computing are being developed very rapidly by commercial companies. These are likely to play an important role in the way in which we do analysis at a distance on an arbitrary hardware platform, using the most up-to-date code and data. The prototype projects using Java which were presented at this workshop [19, 20] already show the immense potential of this approach.

The use of modern technologies alone will not guarantee the success of future HEP software projects. It seems likely that the weakest element of future HEP software projects is neither the software techology nor the software engineering methodology, but rather the personnel involved. Good physicists work creatively (to put it politely) and this can be damaging to a large software effort, which requires rigour and uniformity. We need to accept that not all physicists can be involved in the development of the core physics analysis software. This should be developed by professional software engineers together with a small number of physicists who have undergone training in modern software technologies and methods. To ensure success we need to treat the software of future HEP experiments as seriously as subdetector systems which nowadays are developed by specialised physicists and professional engineers, in close collaboration with industry.

Acknowledgements

We thank the participants and organisers of HEPVIS 96 for such a pleasant and informative conference, and the many colleagues with whom we discussed the frustrations of offline analysis.

References

1 R. Brun and J. Zoll. Zebra user guide. CERN–CN Long Writeup Q100.
2 V. Blobel. The bos system. DESY R1-88-01, 1988.
3 Hbook reference manual. CERN Program Library Long Writeup Y250, 1995.
4 The L3 detector simulation is based on GEANT Version 3.15 (see: R. Brun *et al.*, "GEANT 3", CERN DD/EE/84-1 revised, (1987)). The GHEISHA program is used to simulate hadronic interactions (see: H. Fesefeldt, RWTH Aachen Report PITHA 85/02 (1985)).
5 Particle Data Group, R. M. Barnett *et al. Phys. Rev.*, D54:1, 1996.
6 T. Sjöstrand. Pythia 5.6 and Jetset 7.3 Physics Manual. CERN-TH 6488/92 (Revised), 1992.
7 B. F. L. Ward S. Jadach and Z. Wąs. *Comp. Phys. Comm.*, 79:503, 1994.
8 F. Paige and S. D. Protopopescu. *Brookhaven National Laboratory Report BNL 38034*, 1986.
9 F. James and M. Roos. Minuit. CERN/CN Long Writeup D506, 1981, revised 1989.
10 R. Brun et al. PAW Physics Analysis Workstation. CERN/CN Long Write-Up Q121, 1989. (see also: `http://wwwcn.cern.ch/asdoc/Welcome.html`).
11 I. C. Brock. Mn_Fit – a fitting and plotting package using minuit. L3 Note ***K* 918**, available from the L3 secretariat, CERN, 1991.
12 See, for example: `http://wwwcn.cern.ch/asdoc/WWW/naglib/nagnew.html`.
13 M. Marino and V. Innocente. From SA/SD to OO methods or the Design Problem – a CMS experience. In *Proceedings of the AIHENP'96 Conference*, EPFL-UNIL, Lausanne, Switzerland, September 2-6 1996.
14 D. Duellmann. HEP Data analysis based on an ODBMS store. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS 96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996.
15 Y. Adesanya. A Prototype Data analysis/Visualisation Framework for the LHC era. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS 96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996.
16 S. Giani. GEANT4 R&D: current status and future milestones. In *Proceedings of the AIHENP'96 Conference*, EPFL-UNIL, Lausanne, Switzerland, September 2-6 1996.
17 J. Allison. Graphics for GEANT4. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS 96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996.
18 See: `http://wwwcn1.cern.ch/asd/cernlib/rd45/index.html`.
19 C. Coperchio et al. Java for event display - the WIRED project. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS 96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996.
20 A. Johnson. A Java based Analysis Environment. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS 96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996.