SLAC-PUB-5520
April 1991
(E)

Physics Analysis Tools*

Paul F. Kunz

Stanford Linear Accelerator Center
Stanford University, Stanford CA 94309

1.0 Introduction

There are many tools used in analysis in High Energy Physics (HEP). They range from low level tools such as a programming language to high level such as a detector simulation package. This paper will discuss some aspects of these tools that are directly associated with the process of analyzing HEP data.

Physics analysis tools cover the whole range from the simulation of the interactions of particles to the display and fitting of statistical data. For purposes of this paper, the stages of analysis is broken down to five main stages as shown in Figure 1. The categories are also classified as areas of generation, reconstruction, and analysis. Different detector groups use different terms for these stages thus it is useful to define what is meant by them in this paper.

The particle generation stage is a simulation of the initial interaction, the production of particles, and the decay of the short lived particles. Its output is the 4-vectors of the long lived particles. An example of such an analysis tool is the Lund Monte Carlo[1].

The detector simulation stage simulates the behavior of an event in a detector. It propagates particles through the detector, generates the space points, and writes its output in a form expected from the real detector.

The track reconstruction stage does pattern recognition on the measured or simulated space points, calorimeter information, etc., and reconstructs track segments of the original event. Some form of particle identification is usually also done and summarized as a list of probabilities that a track is a certain kind of particle.

The event reconstruction stage takes the reconstructed tracks, along with particle identification information and assigns masses to produce 4-vectors. It also takes combinations of these 4-vectors to form vertices and the event topology hypotheses. The output of this stage is in the form of histograms and n-tuples.

Finally the display and fit stage displays statistical data accumulated in the preceding stages in the form of histograms, scatter plots, etc. Fitting the data to known functions is also done in this stage.

The remainder of this paper will consider what analysis tools are available today, and what one might expect in the future. In each stage, the integration of the tools with other stages and the portability of the tool will be analyzed.

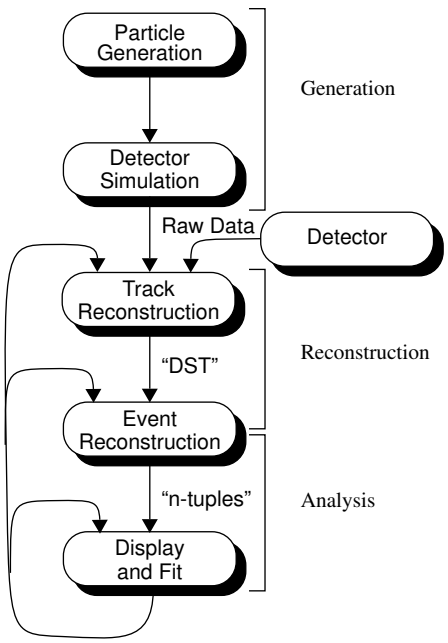

Figure 1. The analysis chain.

* Work supported by Department of Energy, contract DE-AC03-76SF00515.
Invited talk presented at Conference on Computing in High Energy Physics, Tsukuba, Japan, 11-15 March 1991.

2.0 Current Tools

Figure 2 gives a summary of some commonly used tools and the areas they cover. They are grouped by those originating at and/or distributed by three laboratories.

From CERN, there are tools at the top and bottom of the analysis chain. At the top is the LUND Monte Carlo and similar programs[2] which cover the particle generation, while the GEANT package[3] simulates the detector response. At the bottom of the analysis chain is the PAW program. The black arrow indicates that PAW in its native form is limited to display and fitting, while the gray arrow indicates that by using user written code one can extend PAW's capabilities upward towards event reconstruction. Note there is a large gap from where GEANT leaves off and PAW begins.

DESY also has some tools at the bottom of the analysis chain. The GEP package[4] introduced the concept of n-tuples to HEP in the late 1970s. The KAL package[5], although not widely used, is included because it is the only example of an event reconstruction which is in the form of an analysis tool.

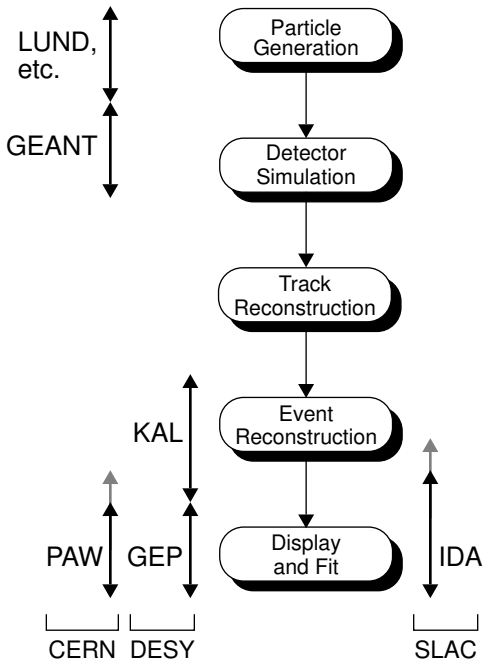

Figure 2. Commonly used analysis tools.

From SLAC, the IDA package[6] also covers the bottom part of the analysis chain. Because its input format allows structured data, its reach is higher than PAW's or GEP's. It too has the capability of user written extension which extends it reach upwards as is shown in gray. Shown is the original IDA, the Mark III version. The SLD version is used as a shell in which the whole analysis chain can be placed.

The remaining part of this section discusses some of these tools in more detail.

2.1 LUND, etc.

It may seem strange to consider a program like the LUND Monte Carlo to be a physics analysis tool. But it is an essential tool with many uses. First of all, in designing a detector, it provides a source of simulated events to study the response of the detector. Then after the detector is built and running, it provides a source of simulated events with known properties to compare with experimental results. In the rest of this paper, where ever we refer to LUND, we also mean most of the other event simulation packages such as ISAJET, etc.

The LUND Monte Carlo and similar programs are not a tool in the sense of a fully packaged interactive program. Rather, it is a package, callable from a user supplied main program. To customize what one wants to simulate, one has two choices. The first is to set some parameters via FORTRAN routines that have been supplied as part of the package. The set of parameters that can be changed this way is rather limited. The other choice is to rewrite some of the physics routines. This is rather painful and difficult because one has to understand the details of how the program and its internal data are organized.

The input to LUND is a random number seed. Since support for random numbers is not part of the FORTRAN language, the LUND package supplies its own random function in which the user can supply the random functions supported at his site. The output of LUND is in one of its FORTRAN COMMON blocks. It's up to the user to copy this data into his preferred means of feeding the data to the next tool in the chain.

The LUND package is written in FORTRAN 77 and doesn't need support from other packages apart from a random number function. It has thus attained a very high degree of portability across many platforms.

One integrates the LUND packages with the rest of the analysis chain in one of two ways. First one can incorporate it as part of a larger program, as discussed above. The other

is to provide a main program to drive it and a set of routines to copy its output to a disk or tape file.

Thus we see that although LUND is an analysis tool in the sense that it is a very important part of the analysis chain, it is not a tool in the sense of a program ready to run with an interactive dialogue with its user.

2.2 GEANT

GEANT is a analysis tool for both detector design and for measuring detector response. However, from the user's point of view, it is much more a *toolkit* than a package. Like LUND, the user supplies the main program driver and the set of output routines. To define the detector geometry, the user supplies calls to a set of routines from the package from his driver program. Essentially any type of volume that is needed to describe a detector can be defined. Once so defined, the user makes a call to generate an event (which he must supply or use LUND), then the GEANT package can propagate the particles in the detector, taking into account the physics processes that will alter the path. However, the user must also supply routines to take the spatial information thus calculated and save it in his desired format.

Designing a detector this way is rather tedious and it usually takes a long time to implement. However, one can simulate detector response to a very high level of detail. The tracking can also consume a large amount of CPU time. Presumably, the generality of the code has forced some trade-offs.

There is no standard detector response output in the GEANT package. The user supplies his own code to record it. However, storing the detector definition to a file is supported in ZEBRA format. Thus, the user integrates GEANT with the rest of his analysis chain by writing custom code.

GEANT itself is written in portable FORTRAN 77 language with few exceptions. However, its portability is limited to those platforms to which its underlying support packages have already been ported. These support packages include CERNLIB, ZEBRA, HBOOK4, and GKS. Fortunately, these packages have been ported to most of the platforms considered for use in HEP.

2.3 PAW: Physics Analysis Workstation

Although the name of this program contains the word *physics*, the PAW program contains no physics. However, PAW is an important tool to visualize physics in the form of histograms and scatter plots as well as to fit these distributions to known functions.

The input to PAW is limited to HBOOK4 files. These include 1D and 2D binned histograms and n-tuples. Thus the common use of PAW is for some external program to do track and/or event reconstruction, with imbedded calls to the HBOOK4 package to generate data to be displayed by PAW. The use of n-tuples to transfer information from the physics analysis packages to PAW is an important aspect of the success of PAW, because the actual histograms can be interactively generated and displayed from data in the n-tuples. However, it is all too frequent that the n-tuples do not contain enough data, so the user must go back to his external physics analysis program to generate another n-tuple set. PAW's inability to accepted an input file with structured event information is thus a serious limitation.

PAW has achieved a high degree of portability to other platforms where its underlying packages have already been ported. This is essentially the same set of CERNLIB packages as GEANT. On most platforms, PAW is command line driven and is considered by most users as awkward. The command lines are parsed with the KUIP package which probably contributes to this problem as well as causing poor consistency between the commands. Development is underway to add an OSF/Motif graphical user interface (GUI) to PAW[7]. In spite of its lack of structured input data and its difficulty of use, PAW is used regularly by perhaps a thousand people.

2.4 KAL: Kinematics Analysis Language

The KAL package was developed for use by the ARGUS collaboration at DESY. To use KAL, one writes a command file in its language and runs the program to produce histograms and n-tuples.

An example command file is shown in Figure 3. The language is like an extension to FORTRAN. However, certain constructs have a lot more power behind them. For example, "**`SELECT K+PI-`**" means to take all pairs of tracks iden-

```
DATA CHMAX 16.0
CUT CHI2VX 64
CUT COSTHETA 0.92
HYPOTH E+ MU+ PI+ 3 K+ PR 1
IDENT PI+ PI-
IDENT K+ K-
SELECT K+ PI-
 IF 1.3 <= M < 2.5 THEN
  FITV0 CHI2 3.
  IF ACCEPT THEN
  SAVE D0=
  ENDIF
 ENDIF
ENDSEL
```

Figure 3. KAL command script example.

tified as K^+ and π^- and calculate vertex parameters. This not only invokes the physics calculation, but also does all the looping and conditional constructs that would be necessary to achieve the same effect in FORTRAN. Also the "**ACCEPT**" construct not only saves the results for reuse in further vertices, but also insures that the input tracks used in this vertex will not be used for another vertex.

KAL is clearly a very powerful physics analysis tool and one could ask why we haven't seen more examples of such tools. KAL itself has not been used outside of the ARGUS collaboration for at least two important reasons. The first is that its input is tied to the ARGUS mini-DST format and the second is that it is also hard wired to the ARGUS particle identification capabilities. This inhibits the reuse of KAL in other environments. That is, to incorporate KAL into the analysis chain of another collaboration, major portions of it would have to be rewritten. This has been done for the ALEPH collaboration in the form of the ALPHA package[8] where KAL is a package of FORTRAN callable routines.

The output of KAL is in the form of n-tuples for use by GEP. Recently output in the form of HBOOK4 files has been added.

3.0 Future Tools

Nobody can predict the future with any certainty, especially in the area of computing technology since it is moving so rapidly in both hardware and software. The key question in the author's mind is whether the future will follow an evolutionary or revolutionary approach. The evolutionary approach would mean the gradual migration to FORTRAN '90 for its significant new features that should ease the task of writing physics analysis codes as well as writing analysis tools[9]. It will also see the addition of a GUI to existing tools. Of course, improvements to performance by using better algorithms and methods are also expected.

A revolutionary approach is starting to take shape amongst a few pioneers in HEP. This approach is characterized by a change of the programming language and the use of the object oriented paradigm. Also, there are analysis tools beginning to emerge that were designed from the start with the GUI in mind.

3.1 The First Tool: Programming

Up to the present time, nearly all of the physics analysis tools have been based on the FORTRAN programming language. It is commonly thought that HEP uses just FORTRAN. However, HEP most frequently uses FORTRAN plus some other package to make up for the limitations of FORTRAN. Examples of such packages are numerous and have been discussed in these proceedings and in previous conferences. GEANT, for example, is written in FORTRAN plus ZEBRA as are many track and event reconstruction programs for large collaborations. The reasons for these additional packages are simple. First there is the need to move data between memory of the running program and disk that FORTRAN alone is very messy at doing. Another reason is the need to handle data, such as tracks and vertices, as structured entities which FORTRAN alone does not support.

The use of such packages has a number of drawbacks. Amongst them are that one might lose access to data by name, one may lose portability of the code if the package hasn't been ported yet, and one may limit the usability of the symbolic debugger. Fortunately, none of the existing packages suffers from all the drawbacks.

FORTRAN also needs help with controlling changes in the source code that may be different for different platforms. This is known as configuration control and it is not supported in the language. Thus in HEP, where portability is desired, one has to use some external system, such as PATCHY[10] or EXPAND[11].

The C programming language is much better than FORTRAN for both data structures and configuration control. Shown in Figure 4 are some segments of C code that one might use in dealing with a track entity. Note the expressive power of the language in that access to variables is by full name. Also, the C language deals directly with the dynamic memory allocation of such structures since the memory allocation functions are part of its standard library. Finally, there's nothing lost in using a symbolic debugger because structures are part of the language, thus known to the existing debugger.

Clearly, C is much better at handling data structures then FORTRAN plus some additional package. Although

```
struct track {
     float px;
     float py;
     float pz;
};
…    …   …
struct track **mctrack;
…    …   …
px = mctrack[it]->px;
```

Figure 4. Segments of C code.

many large collaborations have discussed abandoning FORTRAN, none has done so (yet). One reason they stayed with FORTRAN is reluctance to learn a new language, which is ironic since in each case they had to learn a big system to complement FORTRAN.

C alone, however, doesn't answer the problem of input and output of the data structures. A prototype that does such input and output for C structures is the Cheetah system[12]. It also provides a user friendly set of C functions to manage these structures. In addition, Cheetah writes a full dictionary of the data structures at the head of the file. This dictionary is useful for analysis tools that will read the files. The dictionary contains the names of the structure types, the names of the family of pointers to them, the name, type, size of the members of the structures, and even comment strings on each member. The Cheetah system is also network wise in that it can move data to memory either from a local file or over the network from a remote server.

For all the power and flexibility of Cheetah, the package itself is only about 2,500 lines of C code (about 1/10 the size of ZEBRA). Cheetah makes use of only the C standard library and has been ported to such diverse platforms as UNIX, VMS, and VM without a single line of code change. It is a relatively easy system to learn and was written by a novice C programmer working part time over the course of a year.

Cheetah is just one example of the additional power and flexibility that can be gained by taking a revolutionary approach and changing the base programming language. The C language offers other features as well, such as configuration control. As HEP moves towards greater use of UNIX, one will find that the C language will become increasingly important for dealing with the operating, windowing and networking systems. It has also become the language of choice of many physicists involved with data acquisition systems. Thus, one big question for the future, in the author's opinion, is will FORTRAN '90 be good enough to stop the migration to C?

Many researchers outside of HEP have found the object oriented programming (OOP) approach an even better paradigm for writing simulation and analysis tools. It is a real revolution in constructing programs. The OOP paradigm is not tied to a specific language. The languages of Objective-C, C++, SmallTalk and Eiffel have all been used in various analyses outside of HEP.

The advantages of OOP are many and can not be fully explained in the limited space of this paper[13]. OOP techniques organize both program and data into blocks. OOP inherently does the memory management that needs to be done explicitly with procedural languages. Customizing is done by inheritance techniques whose power should not be underestimated. The polymorphism, which can be used in both the function and data naming space, is also of great value.

In general, the OOP paradigm allows for code that is easier to maintain, expand and modify; all goals of various software engineering techniques and tools. OOP code is also much easier to reuse in different applications. Given the difficulty HEP has had in integration of analysis tools, as described in section 2.0, a revolutionary approach such as OOP may be just what HEP needs.

In the following sections, prototypes of potential revolutionary new physics analysis tools will be described. Figure 5 gives an overview of these tools. These tools cover the whole range from particle generation to display and fit.

The technologies used in each of these tools are OOP techniques and languages. The following sections will describe some highlights of each of these with the exception of the CABS package which is presented elsewhere in these proceedings[14].

3.2 Lund++

Lund++ is an acronym invented by this author for early investigations into using an OOP language to do particle generation simulations[15][16]. The original prototype was

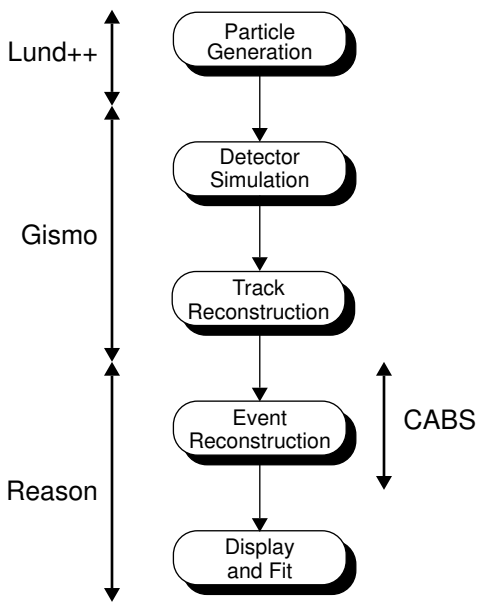

Figure 5. New tools of the early '90s.

written by Richard Blankenbecler at SLAC using the Objective-C language. It was later rewritten and generalized by Leif Lonnblad at SLAC with the C++ language. These prototypes have shown that the OOP approach is extremely powerful for this kind of simulation.

The design of the program revolves around a particle data table (PDT) that is composed of a set of Storage objects. The master Storage object contains a list of particles used in the simulation. For each, the properties such as mass, charge, lifetime, etc. are stored. A key part of the design is that each particle has a pointer to another Storage object which contains a list of branching ratios. In this list, one not only has the branching fractions, but also two pointers; one to a function to calculate the decay, and another to a third type of Storage object. The latter Storage object has for each particle in the decay, an index into the top level Storage object.

The PDT is thus a completely general way of keeping all the needed information for the particle generation. For particles with measured properties, one should be able to upload them from the master database maintained by the Particle Data Group[17]. The PDT can also be used for strings, gluons, quark jets, etc. Thus, in principal, further development could lead to a replacement of the current JETSET, ISAJET, etc. This work was done on a NeXT computer, but it is entirely written in portable C++ with no graphics or other NeXT system dependencies.

Future work on Lund++ is being planned. It will optionally use the Cheetah system to put its result into a file, or it will be integrated into a detector simulation program such as the one described in the next section. At the time of this writing, Lund++ is still an investigation into OOP techniques. It is not an official project of either SLAC or University of Lund.

3.3 Gismo

Gismo is a detector simulation and track reconstruction package described in detail elsewhere in these proceedings[18]. A screen dump of it is shown in Figure 6. Like Lund++, it is an investigation into applying OOP tech-

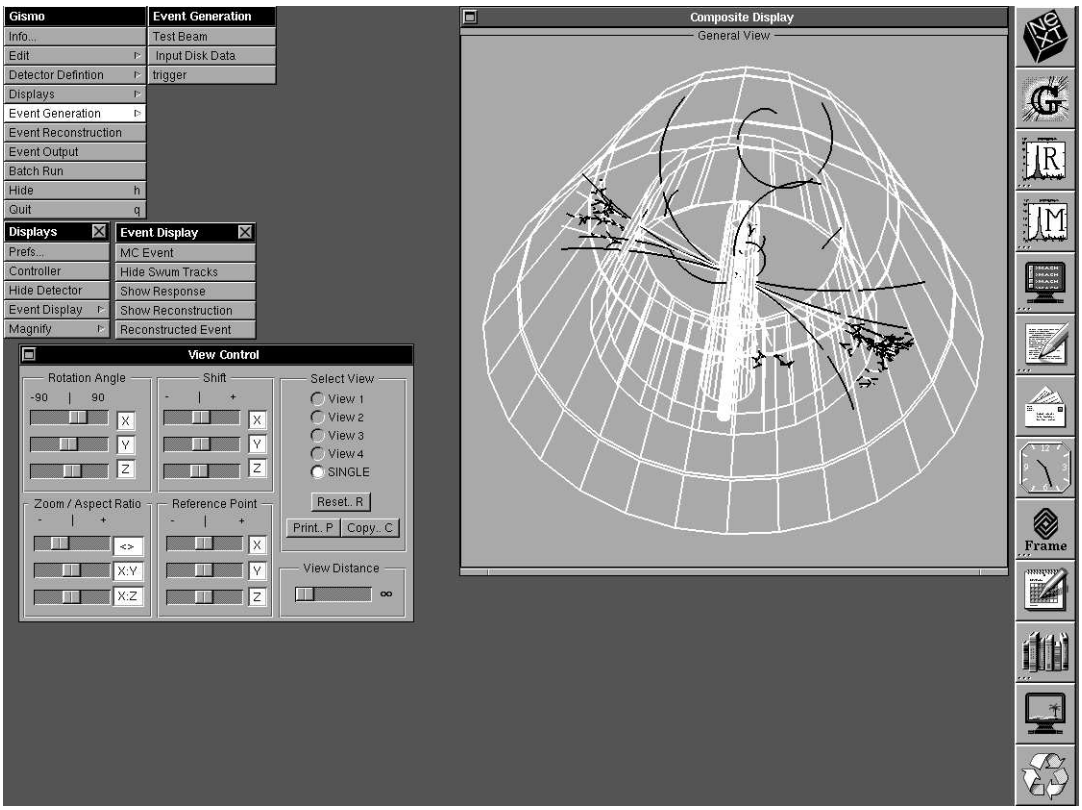

Figure 6. Screen dump of Gismo showing PDT for test beam.

nology to an HEP simulation problem. The results so far look very encouraging and it is far more advanced towards being ready to use then Lund++. It has been developed on NeXT computer, but its computational kernel has no dependencies on the NeXT environment. Its graphics display is PostScript called from Objective-C. Only its user input panels depend on the NextStep environment.

Gismo takes as particle input either its internal test beam object, a Cheetah formatted file, or by incorporating the Lund++ set of objects. The output of Gismo can be the original particles generated, the *swum* tracks, the resultant track segments, and/or the reconstructed tracks. The output subsystem uses Cheetah to write the data to a file for further analysis.

Gismo will also use the same PDT set of objects as Lund++ in order to handle the decay of longer lived particles. This not only demonstrates the reusability of objects, but also a seamless integration particle generation and decay in the detector.

Gismo is more of an object oriented environment for detector design, simulation, and reconstruction then package. It consists of an extendable object oriented toolkit to create a detector design. If the right kind of volume for a new detector doesn't exist, the user can either use an existing kind as a prototype for his own code, or subclass an existing design. The use of OOP makes this much easier than the procedural language based environment of existing packages such as GEANT.

3.4 Reason

The Reason project was started in the summer of 1989 to investigate applying visual programming techniques to physics analysis[19]. The work was done on a NeXT computer, thus the investigators needed to learn OOP technology. From this learning sprang the application of OOP to other areas described in this paper. A screen dump of Reason is show in Figure 7.

An important decision made early on in the Reason project was that its input data format should allow structured data such as a mini-DST used by many collaborations. From this decision, the Cheetah system was born. Unlike the PAW program, n-tuple input to Reason is a subset of

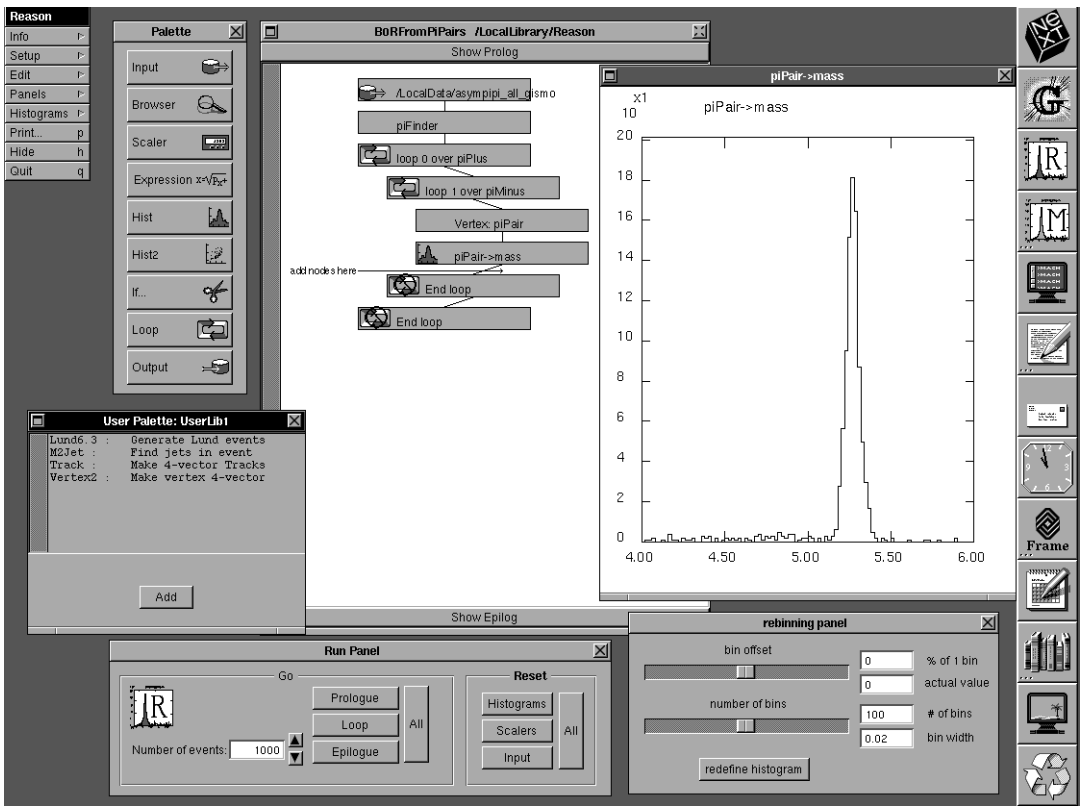

Figure 7. Screen dump with view of Reason.

what it can handle. Using Cheetah format as input has allowed Reason to extend its usability beyond simple display and fitting.

Reason uses OOP not only for its GUI but also for its display and its internal computation kernel. Adding event reconstruction features was thus relatively easy within this OOP framework. For its track and vertex reconstruction, for example, Reason uses the same set of PDT objects as Lund++ and Gismo, demonstrating again the reusability of objects. The output of calculations done with Reason is in the form of Cheetah files or n-tuples.

Because the NextStep environment under which Reason was developed makes the GUI programming so quick and easy, the developers of Reason have concentrated their time in exploring new ways of visualizing the data. The interactive re-binning feature, where a slider is connected to the number of bins used to display a histogram, is but one of many new features found in Reason. It should be noted that by *interactive* in the Reason context, one means that the histogram is constantly re-binned as the slider is dragged with the mouse. This is a new level of interactive computing not seen anywhere else before.

Although usable today to do physics analysis, Reason is still considered a prototype. Work needs to be done in three areas: do some of the obvious display options such as changing the scales from linear to logarithmic, go beyond the obvious such as the interactive re-binning, and to incorporate more physics capability to demonstrate real analysis. A lot of this work will be done in the context of a possible B-Factory at SLAC[20].

3.5 Minuit

MINUIT (spelled with all capital letters) is a well known multiparameter minimization program used in HEP for many years[21]. Its algorithms are well known, but its user interface is not known for ease of use. Minuit (spelled with only initial capital letter) is a NextStep application that uses the same MINUIT program for its computational kernel, but puts a GUI layer for user input and control. A screen dump of Minuit is shown in Figure 8. This appli-

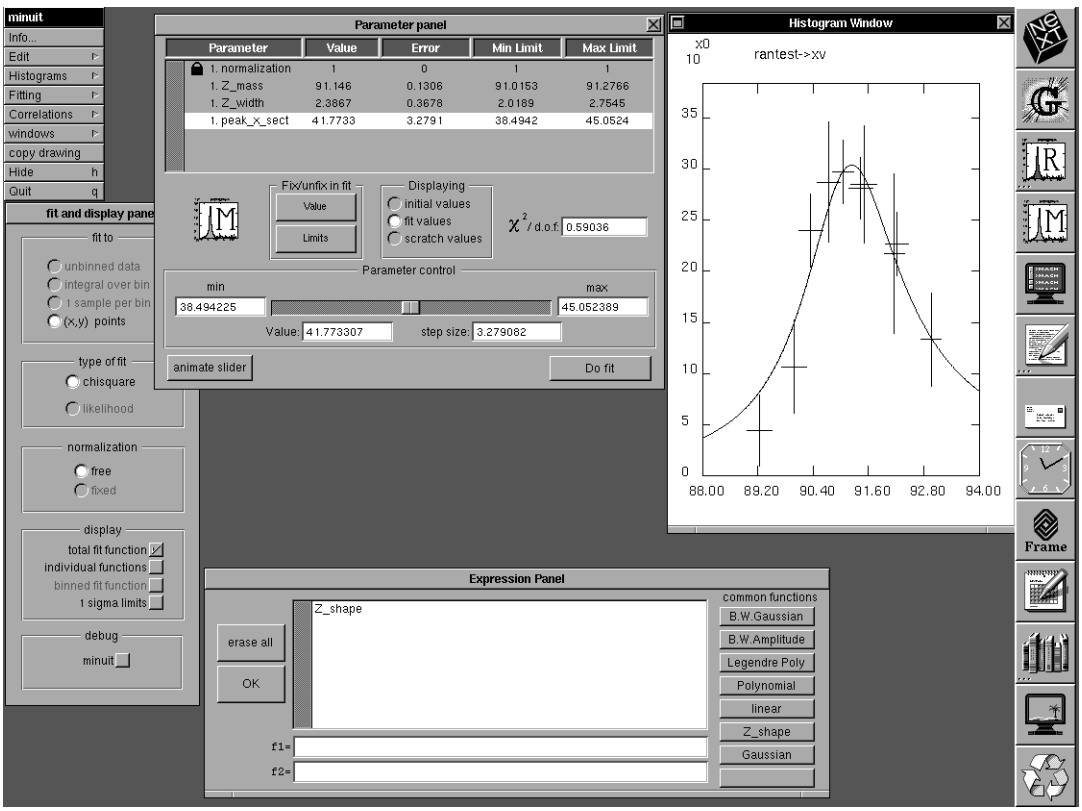

Figure 8. Screen dump with view of Minuit.

cation also uses a package of FORTRAN functions, commonly used for HEP fitting, called BWGNEW[22].

Via Minuit's GUI, the user first selects or writes a function in the expression panel. He then has control over the initial parameters via a scroll-able parameter panel. Each parameter can in turn be connected to a slider to change its value. While the slider is being dragged, the current value of the function is displayed on top of the histogram to be fit. Thus, all guess work is taken out of setting up the initial conditions.

Various types of fits are available including chi-squared or likelihood, binned, non-binned, or integral, etc., are all clearly displayed on a panel of selection buttons. Options that are incompatible are interactively disabled. While a fit is in progress, the values of the function calculated by MINUIT are updated continuously, thus if the fit is not converging, the user can stop it. After the fit is done, the range of the slider is set to $\pm 1\sigma$ so that the user can drag the slider to see the effect of changing the parameters by this range.

The Minuit application uses OOP only for its GUI; the internal calculations are all done in FORTRAN that had been developed previously. Nevertheless, the OOP technology has allowed a truly new user interface to existing code. A new level of interaction with the same code has been achieved.

3.6 Frequently Asked Questions

The people involved in the approaches described in the previous sections are obviously seeking a revolution in the ways of doing physics analysis. There work raises many questions which need some comments.

First of all, it would appear that these people are proposing to rewrite all the analysis programs starting from scratch. However, this is not at all the case. They are using the methods and the algorithms of the past and are simply putting them in a new and more productive environment. They will reuse any existing FORTRAN functions that are sufficiently modular, which generally means functions that do not depend on packages such as ZEBRA.

It would also appear that they are writing in language that precludes them from running their analysis tools on a mainframe. While this may be the case for the moment, it is also true that much production processing today, is and probably to a greater degree in the future, will be done on farms of UNIX workstations where the OOP languages are available.

It would also seem that by basing their GUI work on the NeXT platform one precludes ever running these programs on another machine. Their answer is that for the moment, it is much too hard to port their GUI to a more commonly available window system, such as X11, because of the severe lack of high quality and easy to use development tools. However, they fully expect that this situation will change someday at which point the code could be ported.

One is left for the immediate future, however, with the need to buy a computer with the NextStep environment in order to use these physics analysis tools. NextStep is, of course, available on all computers from NeXT and expected soon to be released on IBM's RISC System/6000 workstations. In the case of NeXT at least, a machine to run any of the tools mentioned in this paper costs as little as US$3,000. Thus for less than the cost of a physicist's secretary's machine for word processing, one can buy a reasonable physics analysis workstation.

4.0 Conclusion

Of the standard physics analysis tools that are in use today, there's a big gap between GEANT and PAW. The only tool that fills that gap is KAL but it is tied to the ARGUS collaboration. In some cases the lack of integration from one tool to another is due to lack of input and output formats that can be used. Although many tools are highly developed, their integration with each other remains poor. However, we can expect these tools to continue to evolve.

The use of the C language and object oriented languages may lead to a revolution in physics analysis tools. Early prototypes cover all the needs from particle generation to display and fit. Development of these prototypes has been amazingly quick. They have also shown what a difference the object oriented paradigm and a good GUI makes in the development environment. Time will tell if these prototypes represent the wave of the future or a passing fad.

5.0 References

[1] T. Sjostrand, Mats Bengtsson, Comput. Phys. Commun. **43** (1987) 367.

[2] A. P. T. Palounek, S. Youssef, Monte Carlo Programs And Other Utilities For High-Energy Physics, LBL-29115-mc (May 1990.)

[3] R. Brun, R. Hagelberg, M. Hansroul, J. C. Lassalle, GEANT: Simulation Program for Particle Physics Experiments. User Guide and Reference Manual, CERN-DD/78/2 Rev. (July 1978.)

[4] E. Bassler, Comput. Physics Commun. **45** (1987) 201.

[5] Hartwick Albrecht (DESY), private communication.

[6] T. H. Burnett, Comput. Physics Commun. **45** (1987) 195.

[7] R. Brun, Proc. Computing in High Energy Physics, Tsukuba (March 1991.)

[8] H. Albrecht and F. Blucher, ALPHA User's Guide, ALEPH 89-151 (September 1989.)

[9] M. Metcalf,Proc. Computing in High Energy Physics, Tsukuba (March 1991.)

[10] H. J. Klein, J. Zohl, PATCHY: Reference Manual Revised For Version 4.09, CERN-23 (October 1983.)

[11] The CDF Collaboration, private communication.

[12] P. Kunz and G. Word, Proc. of the Workshop on Data Structures for Particle Physics Experiments, Erice (November 1990.)

[13] P. Kunz, Proc. Computing for High Luminosity and High Intensity Facilities, Santa Fe (April 1990.)

[14] N. Katayama, Proc. Computing in High Energy Physics, Tsukuba (March 1991.)

[15] W. B. Atwood, et al, Proc. Symposium on Detector Research and Development for the Supercollider, Fort Worth (October 1990.)

[16] L. Lonnblad, University of Lund, private communication.

[17] Particle Data Group, Physics Letters **B 239**, April 1990.

[18] W. B. Atwood, T. H. Burnett et al, Proc. Computing in High Energy Physics, Tsukuba (March 1991.)

[19] W. B. Atwood et al, Proc. Computing for High Luminosity and High Intensity Facilities, Santa Fe (April 1990.)

[20] T. Glanzman, Proc. Computing in High Energy Physics, Tsukuba (March 1991.)

[21] F. James, M. Roos, Compu. Physics Commun. **10** (1975) 343.

[22] W. Lockman, BWGNEW 2.0 User's Guide, SCIPP 89/08 (March 1989.)