SLAC–PUB–5242
April 1990
(T/E)

THE REASON PROJECT*

William Atwood†, Richard Blankenbecler,
Paul F. Kunz, Benoit Mours‡, Andrew Weir
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94309

Gary Word
Vanderbilt University
Nashville TN 37235-1807

ABSTRACT

Reason is a software package to allow one to do physics analysis with the look and feel of the Apple Macintosh. It was implemented on a NeXT computer which does not yet support the standard HEP packages for graphics and histogramming. This paper will review our experiences and the program.

INTRODUCTION

The goal of the Reason project is to develop a software package to allow one to do a physics analysis with the look and feel of the Apple Macintosh. The authors feel that the Macintosh has set the standards for a user friendly interface with its pop-up menus, mouse driven interaction, and little need for users to consult manuals. The Reason project is a three part experiment. The first experiment is an attempt to achieve the look and feel of the Macintosh for physics analysis. The second is to see if physicists would like to do their analysis this way. And finally, Reason is an exploration into the world of UNIX and its tools, which are not commonly used in high energy physics.

THE CHOICE OF A NeXT COMPUTER

The target of the Reason project is the analysis of summary physics data where the storage and CPU requirements are well matched to affordable workstations. In considering the needs for doing this kind of analysis, the authors considered the following list of desirable attributes:

- Enough CPU performance to do simple physics analyses interactively.
- Enough memory to hold the program and data.
- High I/O throughput, since data will be frequently read from disk.

* Work supported by the Department of Energy, contract DE-AC03-76SF00515.
† Currently at CERN, Geneva.
‡ Permanent Address: LAPP, Annecy.
Copyright (c) 1990 The Board of Trustees of The Leland Stanford Junior University. All Rights Reserved.

Invited paper presented at Rencontres de Moriond, Les Arcs 1600, France, March 12-16, 1990, contributed paper presented at International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics, Lyon Villeurbanne, France, March 19-24, 1990, and contributed paper presented at Computing in High Energy Physics, Santa Fe, N.M., April 9-13, 1990.

- A large amount of disk storage to hold the data sets.
- Good networking capabilities in order to transfer data sets from mainframe computers where they are most likely to be produced.
- Megapixel graphics for display of results and user interface.
- A good programming environment to develop analysis code; generally one thinks of a good FORTRAN environment.
- A good programming environment to develop the Graphical User Interface (GUI).

A number of workstation platforms were considered with these features in mind. Clearly, if one wanted to achieve the look and feel of a Macintosh, then it would have seemed logical to choose a Macintosh computer to be the platform. However, the authors felt that the Macintosh was weak in the areas of memory space, I/O throughput, disk space, networking, the standard display, and the GUI programming environment. UNIX workstations, such as an Apollo, SUN, or Silicon Graphics were also considered. These workstations are strong contenders except in the area of the GUI, where we felt they were just as weak, if not weaker, than the Macintosh. A VAX/VMS workstation was also considered, especially since it has a well known FORTRAN environment. However, it also was poor in the area of developing the GUI. Only a NeXT computer with its unique GUI, introduced in the fall of 1988, seemed to match our requirements.

Although weak on CPU performance compared to RISC UNIX workstations, the NeXT was at least equal to other UNIX or VAX/VMS workstations in other areas of hardware capabilities. Its main hardware attributes are listed in Table I. Only one requirement was initially missing from the NeXT computer: a FORTRAN compiler. Its absence led the authors to consider doing physics analysis without FORTRAN and a discussion of our conclusions is treated in a separate section below. A very important feature of the NeXT, however, is the object oriented environment for developing the GUI: NextStep. It is this feature along with the power of a UNIX workstation that led the authors to choose a NeXT as the best platform for the project.

The NeXT is rather a strange computer. Some people consider it as competitor to the Macintosh, while others consider it as a UNIX workstation. When compared to a Macintosh, one generally looks at the Macintosh-style software that is available. There are many useful software packages that come bundled with each NeXT system. Having

Table I. Main attributes of a NeXT workstation

CPU	25MHz 68030, 25MHz, 68882, which is about 4 MIPS.
Memory	8 to 16 MBytes real memory, and virtual memory operating system.
I/O	DMA 4.8MB/s burst rate, 1.4MB/s sustained.
Disk	330 or 660 MB hard disk and 256MB read/write removable optical disk.
Networking	Ethernet (thin-net), TCP/IP protocol suite with 100 KB/s ASCII, 160 KB/s file transfer rate.
Graphics	PostScript
Analysis programming environment	C, Objective-C.
GUI programming environment	NextStep, an object oriented environment.
Cost (with academic discount)	model with 330MB disk, 8MB RAM $8,500 model with 660MB disk, 16MB RAM $12,000

these applications on the same workstation that one uses for physics analysis certainly adds to the desirability of the NeXT platform. Notable amongst the bundled applications are TeX including a previewer, Mathematica, WriteNow (a word processor), Sybase SQL database Server, Webster dictionary and thesaurus, a postscript previewer, a mail program that allows inclusion of voice and graphics, and the Digital Librarian which does a key word search into all the NeXT and UNIX documentation. At the time of this writing, the number of third party applications, the so-called “shrink-wrapped software”, is still very small compared to a Macintosh, but large compared to other UNIX workstations. The quality of some of the available applications is quite high. The desktop publishing package, FrameMaker for example, was used to typeset this article.

To develop Reason, we needed a good programming environment, so the software tools and utilities that come bundled the NeXT were far more important than the shrink-wrap style software. Of prime importance is the fact that the NeXT is a UNIX machine. In detail, it is MACH UNIX which is BSD 4.3 compatible. What this means for doing physics analysis is that the operating system is multi-tasking, multi-user, and has virtual memory. It also means that standard UNIX utilities are bundled with the system such as the complete TCP/IP networking protocol suite (FTP, TELNET, TN3270, etc.), NFS and Yellow Pages for file sharing across the network, EMACS and VI for editing with remote logon, a full screen symbolic debugger, RCS and Make for source code and module management, and hundreds of additional tools, not all of which we used, nor even know about. In addition, the NeXT system has its own set of utilities which enhance the programmer's environment. The mouse-based editor, for example, has keyboard driven commands that mimic the well known EMACS editor. It also has an interesting set of mouse driven features which are designed to enhance its usefulness when editing programs.

Of prime importance is the application called the Interface Builder. This program is used to layout the various windows and panels that are part of the Reason programs. Since the NextStep environment is object oriented, one is really laying out graphical objects with Interface Builder and connecting these graphical objects to the objects represented by the code we have written. This needs to be contrasted with the window layout applications one typically finds on other systems that merely draw pretty pictures and then generate C code to reproduce those pictures in a difficult GUI programing environment.

LIFE WITHOUT FORTRAN

Before choosing a NeXT computer, the authors had considered the consequences of not having a FORTRAN compiler with the system. In other words, what would it really mean not to have FORTRAN in the physics analysis environment, since FORTRAN is generally considered mandatory in the purchase of any HEP computer. The first thing we realized is that without FORTRAN we would not have any histogram package such as SLAC's HandyPak or CERN's HBOOK. Since histogramming is so fundamental in doing physics analysis, the lack of such a package was a potentially fatal flaw in the system. Upon closer inspection, however, we saw that such packages have two parts: the definition and accumulation of histograms, and their display. The first part is relatively simple and we decided we could re-write this part using the C language without much difficulty. The second part, the displaying, is a much larger body of code but it depends on the graphics package. Without FORTRAN meant that one was without the FORTRAN callable graphics packages such as SLAC's UGS, or the GKS package which has become the standard CERN package. The NeXT is supplied with a library of C-callable routines; one routine corresponding to each PostScript language operator. Since graphics is a fundamental part of the program we were about to write, it made much more sense to re-write all the graphics we needed in PostScript than to spend time trying to port a FORTRAN-callable package to the NeXT environment.

Being without FORTRAN meant that we were also without a data management package, such as DESY's BOS, SLAC's JAZELLE, or CERN's ZEBRA, which are being frequently used to organize store the data sets in memory as well as on disk. However, since we would be writing our new code in C, and C already has the facilities to manage data as part of the language (namely structures), those FORTRAN packages are not needed for storing data in memory. For storing the data on disk and for reading and writing the data from disk to memory, however, we needed to write our own package. This new data management package is called Cheetah. Finally, being without FORTRAN meant that we would not have some other utilities that we had become familiar with, that are written in FORTRAN. One example is that of source language pre-processors such as SLAC's MORTRAN. Others are source code management system such as SLAC's DUCS or CERN's CMZ. In all cases, we found native UNIX utilities to replace our old FORTRAN based packages, which were usually more mature and easier to use anyway.

In summary, we realized that FORTRAN was not necessary for the kinds of packages described above. In fact, since all these packages were going to be fundamental to our Reason programs, it was far better to get a fresh start on them, than to spend time porting old code, with its batch-orientation. In practice, we found we needed to extend the features of these types of packages, which meant it was valuable to us to have our own newly written code. We also found it quite easy to make these extensions in the C language, while it would have been more difficult in the FORTRAN language.

Every member of the Reason development team came from a mainframe-based FORTRAN environment, without any previous experience in either C or UNIX. Thus all the team members had a lot of learning to do, and this learning can be considered part of the experiment with the UNIX environment. It is thus worthy of summarizing what it was like:

- C: No one in the Reason team found C difficult to learn. We used Kernighan & Ritchie's book[1] (which is the book that defined the language) and found it quite adequate for people with prior experience with FORTRAN programming. Most of the team would now prefer to program in C than in FORTRAN for all our programming tasks.

- Objective-C: Since the NextStep toolkit is written in Objective-C, we also had to learn object oriented programming. We found a relatively high threshold to learn these techniques, but once learned we found it quite easy to use. The cause of the high threshold is that the concepts are so different from traditional languages such as FORTRAN. However, once learned, programming in Objective-C was not difficult at all.

- NextStep: Once Objective-C was understood, learning NextStep was not difficult at all. In fact, with the Interface Builder application available to build our GUI, it was extremely quick and easy to build our applications.
- PostScript: We were pleasantly surprised to find that PostScript, when used as a set of C-callable graphic functions, was an easy to learn and use 2-D graphics package. It has all the utility of the FORTRAN based packages we had previously used and much more.
- UNIX: It would be unfair to say that all members of the Reason development team truly experienced the UNIX environment because the NeXT graphical user interface to UNIX protects users from many aspects of UNIX. However, one member of the team, the principal author of Cheetah, did all his work from his home by logging into a NeXT at 2400 baud. Thus he saw a pure UNIX environment, without aid of a GUI, and was able to learn the environment and accomplish his tasks with few problems.

Although there was a lot to learn, members of the development team felt comfortable with the adage: ***"if you have to learn something, learn something useful."***

For some other standard HEP packages, however, it would not be practical to rewrite in C. For these packages, we have obtained a FORTRAN compiler from a third party vendor, Absoft. With this compiler, we have incorporated such standard FORTRAN packages as the LUND Monte Carlo, the MINUIT minimization program, and code from some of SLAC's detector groups.

REASON - THE HISTOGRAMMER

Reason is an application which allows the user to perform sophisticated analyses of high energy physics data in an interactive, user-friendly environment. All aspects of the analysis are performed in a highly graphical and intuitive way, by means of input from the mouse and keyboard. The user is able to completely specify the analysis, including input files, cuts, loops over tracks, arbitrary logical or numerical expressions, histograms etc. by making selections from a palette of objects. The application can accomodate user code to perform specific operations, e.g. jet finding, which allow it to be used in many different situations.

The application was written on the NeXT computer mainly in Objective-C, with some parts written in C and FORTRAN, and makes extensive use of the many features of the NextStep computing environment.

Specific features of interest:

The application uses object-oriented programming techniques throughout to achieve flexibility, modularity, simplicity of coding and reusability of code. The graphical user interface was written using the NeXT program "Interface Builder", which allowed rapid prototyping of many of the graphical elements of the application.

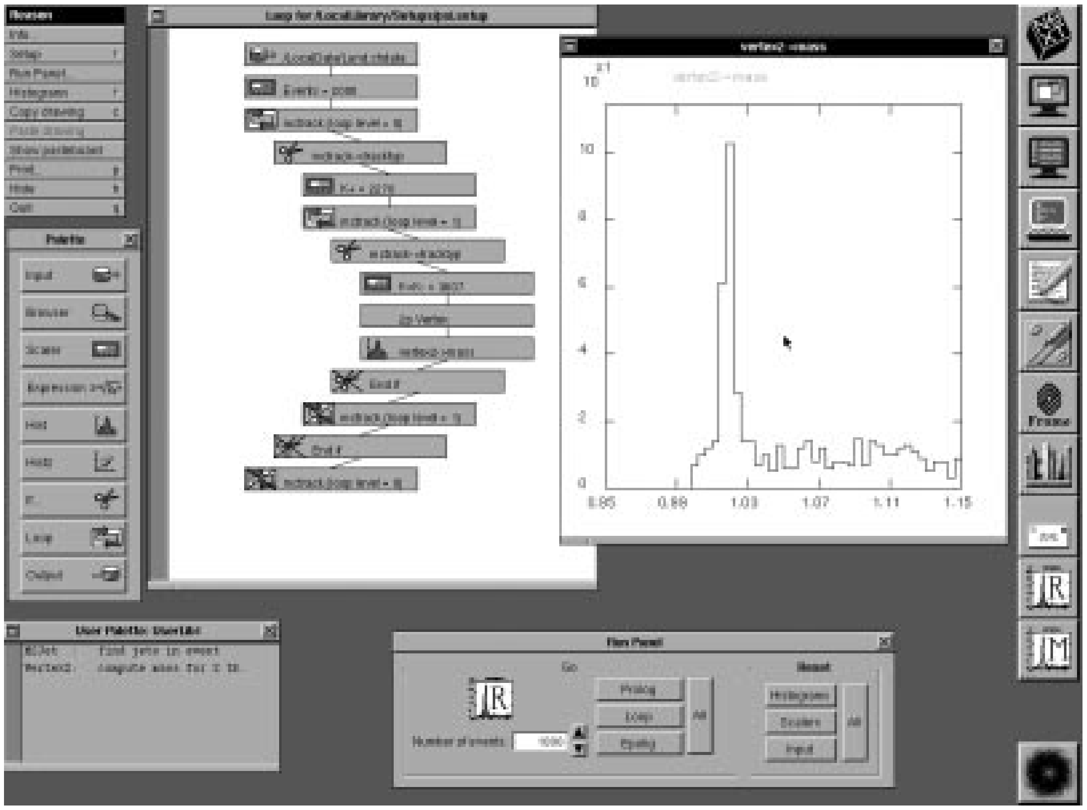

Fig. 1. Screen dump showing Reason - the histogrammer.

Because the user is able to specify the analysis chain in a graphical, as opposed to a lexical format (i.e. a FORTRAN or C program), the possibility of coding bugs and syntax errors is greatly reduced. This enables the user to concentrate on the important aspects of the analysis without wasting time on frustrating debugging of code.

The data is readily accessible by means of a “data browser”; a window which allows the user to examine the values of variables within events, including structures such as individual tracks, vertices etc. An important aspect of the Cheetah data format that is used as input is that it allows structure within an event.

Specific features have been incorporated which make the application “user friendly”. These include the ability to redefine the features of a histogram (e.g. bin width, x and y scales) without needing to rerun the analysis; recalling definitions for cuts, expressions, histograms etc. by “double-clicking” on the relevant graphical object; error checking which brings up the definition panel for objects with illegal definitions to allow the user to make the necessary corrections; multiple histograms allow the user to use one histogram definition for many histograms, e.g. in making a distinct histogram of the momentum of particle tracks for each particle type. The histogram object checks to see if a histogram has been defined for the current track. If it has been defined, it accumulates the histogram. If not, it defines the histogram according to the specified definition and then accumulates it.

Sophisticated fits to the data are made possible by an interface to the fitting application which uses the CERN program MINUIT as the fitting engine.

Program Structure:

The application is structured around the concept of Objective-C “objects” which perform specific functions within the analysis chain. Examples of such objects are histograms, do-loops and if-thens (cuts). When the analysis is to be performed, each object in the chain receives the message “go” in the order specified by its position in the chain. The first object is usually the “Input” object, which, on receipt of the “go” message inputs the next event. Subsequent objects then make cuts on the data, accumulate histograms etc. until the end of the chain is reached. The procedure is repeated for the number of events specified by the user. The user is able to update histograms while the analysis is being performed, or to stop the analysis at any time.

Several objects are reused many times within the application in different contexts. An example of a heavily used object is the “Inspector” which provides a window in which data values for a given event are displayed. In the definition of a histogram, for example, it is necessary for the user to specify which variable is to be histogrammed. The “Inspector” window appears within the histogram definition panel to enable the user to choose the variable to be histogrammed by mouse-clicking on the relevant entry. Similarly, the object which performs cuts on the data uses the inspector window within its definition panel to enable the user to specify which variable is to be cut upon. When the analysis is being performed, the object which uses the inspector sends a message to the inspector object asking for value of the selected variable for the given event. The returned value can then be cut upon, histogrammed, or whatever operation the calling object requires.

It is possible for the user to interface his/her own code with the application in cases when user-specific operations need to be performed on the data. An example of such a “User Object” is a jet-finder, which takes tracks from the event and finds jets according to a specified algorithm. We interfaced the event analysis package VECSUB (which, incidentally, is written in FORTRAN) within the application by creating an object called “Jet”. When this object is initially placed in the chain by the user it creates a data structure called “Jets” into which the output of the jet finder will be placed at run time. Variables within this structure are accessible to all subsequent objects in the analysis chain, including the “Output” object which writes events to a specified disk file. On receipt of the “go” message the Jet object packs the tracks into its FORTRAN COMMON blocks, finds jets, and returns the variables of interest into the “Jets” data structure. By interfacing objects in this way, the user can build upon the basic set of objects to provide an application tailored to his/her needs.

Description of operation:

An example of a session using Reason is shown in Figure 1. The main windows which can be seen are (in clockwise direction from upper left) the main application menu panel, the "Setup window" which contains the analysis chain, a Histogram window, the "Run panel" which controls the number of events to be analyzed, the "User palette" which contains objects written by the user to accomodate operations specific to his/her interests (e.g. a Jet finding object), and finally the Object Palette.

To create an analysis chain, or “Setup”, the user selects the “Setup” item from the main menu (upper left menu in Figure 1.), and then “New setup”. The analysis chain can be constructed by mouse-clicking on the following objects in the Object Palette (seen in the center-left in Fig 1.)

- Input: this is usually the first object in any analysis chain. It allows the user to specify the source of the events to be analyzed. Files of events in Cheetah format* can be read in, or an event generator, such as the LUND Monte Carlo can be used to generate events.
- Browser: this object does not perform any operation on the data, but provides a window with which the user can examine the data within a given event. It is possible to “browse” through the data, examining the available data structures and the values of variables within these structures.
- Scaler: this object simply counts the number of times it has been called. It can be reset to zero at the beginning of each event, if desired. It is useful, for example, for counting the number of events which have been read in, or the number of iterations around a do-loop. This data is available to other objects since the scaler creates a data structure in which it places its current value.
- Expression: this object enables the user to define arbitrary numerical or logical expressions which can be constructed from variables within the data structures. A definition panel appears when this item is selected in which the user types an expression using either C or FORTRAN language syntax. Data values can be easily included by means of the “Inspector” window which is part of the expression definition panel. Expressions can include any standard C function, or even other expressions. A new data structure is created which contains the resultant value of the expression.
- Hist: this object provides the user with one-dimensional histograms. When this object is selected from the Palette, a definition panel is produced in which the user specifies the title, lower limit, upper limit and bin-size for the histogram. The variable to be histogrammed is selected from the “Inspector” window within the definition panel. This variable can be from any available data structure within the event, including scalers, expressions, cuts or other user-defined objects. It is possible for the user to make distinct histograms based on a single definition, e.g. residuals for hits on each wire in a drift chamber. There are several display options including points with error bars, joined points etc. Histograms can be saved to disk files, or be fit using a quick and dirty fitter, or the sophisticated MINUIT driven fitter.
- Hist2: in analogy with the “Hist” object, this object provides two-dimensional histograms. Two inspector windows appear in the definition panel, one to define the x-variable, and one for the y-variable. Possible display options include levels of gray-scale which are proportional to the number of entries in given bins, scatter-plots, or simply the number of events in each bin.

* A utility for converting files in other formats to Cheetah format is available.

- If...: in analogy with the “If...then” statement in C or FORTRAN, this object allows the user to specify the conditions under which certain objects should be called. The definition panel contains an “Inspector” window which selects the variable to be cut upon. There are various conditions under which the “cut” is satisfied, including x > value, x < value, x = value, x != value, low_value < x < high_value and (x < low_value or x > high_value) where x is the selected variable and value, low_value and high_value are specified by the user.
- Loop: in analogy with the “do-loop” of C or FORTRAN, this object allows the analysis chain to loop over structures within the event (e.g. tracks, jets). For nested loops, it is possible for the inner loop to start looping at the outer loop index, if required. The loop index is available to objects within the loop to enable, for example, the calculation of the invariant mass of two particles whose indices are the loop indices of two nested loops.
- Output: this object writes out events to disk files, including all the data structures within the events. It is possible to specify the data structures which are written out, and any new structures defined by objects such as the expression object can also be written. Output files can be concatenated.

Note that none of the objects in the palette described above are related only to physics. That is, these objects are completely general and can be used with any kind of data that has a record structure. The physics objects are contained in a User Palette which is shown in the lower left of Figure 1. In this way, Reason can be customized for different applications.

REASON - THE FITTER

The second part of the Reason project is currently a separate application which performs fits to the data in one-dimensional histograms. This application is a graphical interface with the CERN program MINUIT (7800 lines of FORTRAN) which has been widely used within the HEP community over the last 20 years. A screen dump of the MINUIT application is shown in Figure 2. Since this application and the main Reason application share the same histogramming package, it is straightforward to export a histogram file from Reason into the fitting package and back.

The fitting application attempts to make the process of fitting functions to data more interactive and graphical. The traditional approach requires the user to specify the function to be fit in the form of a FORTRAN function, the fit parameters being specified in a “runcard” file read in by the MINUIT program. The output of the fit was generally in the form of columns of numbers, although the user could interface the program with his/her favorite graphics package to provide graphical output to a terminal or printer.

Our approach involves drawing the function on the histogram in a re-sizable window on the screen. Each of the parameters of the fit can, in turn, be connected to a slider, and the user can interactively redraw the function on the screen by moving the slider. The chi-squared per degree of freedom interactively updates as the slider is moved. This approach enables the user to examine the extent to which the function depends on the parameter in question. The user can, by adjusting each parameter in turn, usually get

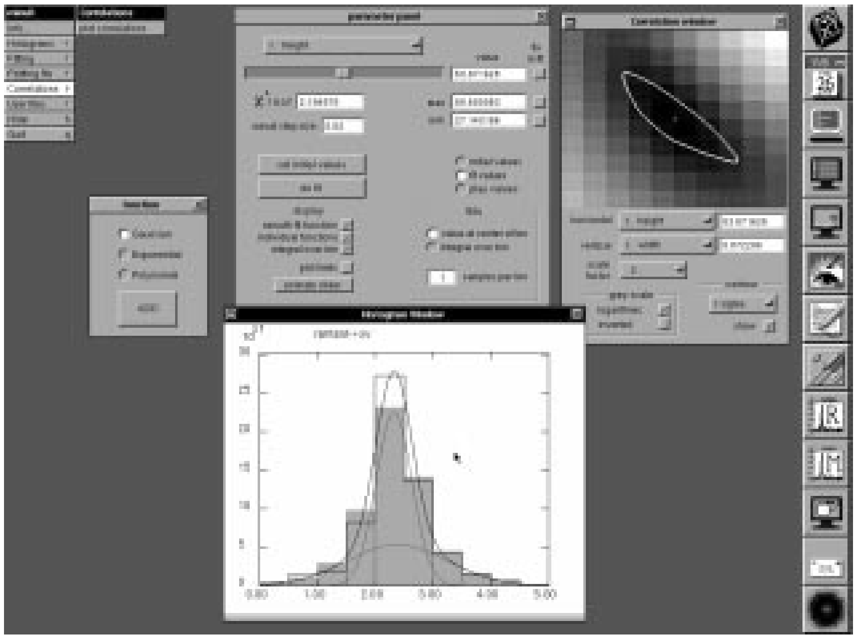

Fig. 2. Screen dump showing Reason - the fitter.

very close to the optimal solution. This simplifies the job of the main MINUIT routines, and minimizes the possibility of convergence to an incorrect solution.

After the MINUIT routines have converged to a solution it is possible to display the errors on each fit parameter in various ways, including shaded regions on either side of the curve and by animating the slider between the plus and minus 1-sigma limits. It is also possible to plot the correlation between any two of the parameters in a two-dimensional gray-scale plot in which the density of the gray at a given x-y point represents the value of the chi-squared when the two parameters have the values x and y, respectively.

In cases where the function changes rapidly across individual bins, the fit can be performed to the integral of the function over the bin, and the integral can be displayed as a transparent gray-scale histogram, superimposed on the data.

The exact form of the fit function is currently specified by the user selecting a function from several simple available functions. These simple functions can be added together to form linear combinations (e.g. gaussian on a polynomial background). We are currently implementing a scheme in which the user can define the fit function using the same expression object as in the main Reason application. This will enable the user to interactively define his/her own fit function, which will be able to be saved in the histogram file, if desired. These defined functions could also become part of a reusable library of function objects.

A screen dump of a typical session is shown in Figure 2. The slider which is connected to one of the parameters can be seen in the "parameter panel" in the upper central region. In the histogram window can be seen a fit containing two gaussians (shown individually as gray curves, and their sum as the black curve), and the integral of the fit function shown as the gray scale histogram. In this example, the fit was performed to the value of the fit function at the center of the histogram bins. Clearly, the fit should have been performed to the integral of the function over the bins.

We also plan to incorporate the full functionality of the SLAC package BWG,[2] which has been used for several years by the Mark III collaboration to perform sophisticated fits to coherent, multiparticle decays of various charmed and strange mesons.

CONCLUSIONS

The Reason project has succeeded in developing prototype applications to do physics analysis with the “look and feel” of the Apple Macintosh. It has been tested using a number of different data sets. It has even been used to re-do the analysis for one of the author’s thesis. In writing these applications, the authors needed to learn programming languages and tools not commonly used in high energy physics and they found that these tools form a very good environment for program development. In fact, the authors are quite surprised in how much has been accomplished in the first nine months of the project. The Reason applications are a product of these very good tools in the hands of a few physicists.

There is currently much work that yet needs to be done before Reason is a fully functional application to do all sorts of physics analysis. Some of this work will be rather routine, like completing the histogram package for all types of display options that are available in mature packages. Other work will be more challenging, such as providing all the visual programming objects to do sophisticated analyses. Reason is perhaps the first example of the NeXT Generation of physics analysis software.

ACKNOWLEDGMENTS

The authors would like to thank Charles Prescott for his support, encouragement and financing of the Reason project. The authors would also like to thank Mike Fero (of MIT), Tom Markiewicz, and Leon Rochester for the encouragement and advice. If it were not for other commitments, each of them would have had an active role in developing the software.

REFERENCES

1 B. Kernighan and D. Ritchie, *The C Programming Language,* Second Edition, Prentice Hall.
2 William Lockman, *BWGNEW 2.0 User’s Guide*, SCIPP 89/08, March 1989.