CERN-Data Handling Division
DD/88/5
March 1988

Development of a large graphics-based application package.

Carlo E. Vandoni

Data Handling Division
CERN
1211 Geneva 23 SWITZERLAND

Abstract

CERN, the European Laboratory for Particle Physics, is one of the largest scientific research laboratories in the world. Most of the research work carried out in the Laboratory is not only strongly dependent upon the large particle accelerators in operation, but it also relies heavily on the use of computer systems of which several hundreds, of various types, are presently installed. During the last twenty years, CERN has played a leading role as the focus for development of packages and software libraries to solve problems related to High Energy Physics (HEP). The results of the integration of resources from many different Laboratories can be expressed in several millions lines of code written at CERN during this period of time, used at CERN and distributed to collaborating Laboratories. Nowadays, this role of software developer is considered very important by the entire HEP community.

This paper reports the development of a large application package. The paper is divided into two parts. Firstly, a large software package where graphics play a key role (PAW - Physics Analysis Workstation) is taken as a case study, and its origin and main characteristics are presented in detail. PAW is essentially an interactive system which includes many different software tools, strongly oriented towards data analysis and data presentation. Secondly, the criteria and rules to be obeyed for the development of a large package in the environment of a large international collaboration are outlined. Then, the development process of the package is described and some of the problems encountered and the solutions adopted are discussed.

Key words: computer graphics, high energy physics, computer graphics applications, software packages development, modularity, portability.

1. Introduction

CERN, the European Laboratory for Particle Physics, operated by the European Organization for Nuclear Research, one of the largest scientific research laboratories in the world, has as its primary function to provide European physicists with world-class particle physics research facilities which could not otherwise be obtained with the resources of the individual countries. This makes CERN the principal European centre for fundamental research in the structure of matter. The field of research is variously known as "particle physics" (since it is the study of the behaviour of the smallest particles of matter), "subnuclear physics" (since the particles involved are on a smaller scale than the atomic nucleus) or "high energy physics" (since high energies are needed to perform the research). It is "pure research" and is not concerned with the development of nuclear power or armaments. Most of the research work carried out in the Laboratory is strongly dependent upon the use of large particle accelerators, and computer systems of which more than 300, of various types, are presently installed. This multi-mainframe and multi-minicomputer complex is available daily to over 4,000 users, who are essentially physicists working on-site on various physics experiments.

In parallel with the physics research, development work is carried on continuously to improve the accelerators, detection systems, computing facilities and other auxiliary installations. The range of equipment available at CERN is amongst the finest assembled for pure physics research at any site in the world. Fourteen European countries contribute, in proportion to their net national income, to the CERN budget which was 722 million Swiss francs in 1987. The present number of staff members is approximately 3500 and, in addition, there are some 4300 scientists, engineers and technicians from 190 universities and research institutes, mainly from the CERN Member States.

2. Computing at CERN

The development of experimental techniques in high energy physics over the past 30 years has leaned heavily on the parallel development of electronic computers. In a typical experiment, hundreds of thousands, or even millions, of particle events are measured by using suitable detectors to observe the products of collisions between particles. After an event has been measured, its analysis often involves long and complex calculations. The need to repeat these calculations for so many events gives rise not only to a huge processing load but also to enormous data handling requirements. Computers entered the field of high energy physics very early on, and they are now so pervasive that there is almost no single building on the CERN site (there are hundreds of them) where at least one computer is not present. In addition, the type of research carried on here has such unusual requirements for both computer hardware and computer software that it is often impossible to find suitable products on the market, and this has led to several locally-developed hardware devices and software packages.

2.1 Use of computers and computer graphics

Computer graphics is just one important aspect of the computing facilities present on the site. In fact, computers are used at CERN for three main purposes [1] and [2]:

2.1.1 Control of the particle accelerators installed on the site

This application (control of accelerators) is a typical process control application. Nowadays, particle accelerators are controlled from central control rooms via networks of literally hundreds of computers. Here computer graphics are very important, as the parameters of the machine are controlled via graphic displays, and data on the status of the accelerators are distributed everywhere in the experimental areas via dedicated closed-circuit television networks.

2.1.2 Control of the experimental apparatus and data taking equipment

In the past, the main types of detectors were bubble and spark chambers, producing their output on photographic films which were analyzed off-line in a manual or semi-automatic way. Nowadays, almost all detectors produce output in the form of digital signals, and computers are used at all experiments to control the experimental apparatus, to perform on-line data reduction and to record data on magnetic tape for further analysis. In fact, the data-taking computer is very often linked to extremely rapid electronics (locally-developed) for on-line decision and data reduction. Graphics devices are always part of the standard configuration of the experiment's control computer.

2.1.3 Analysis of the experimental data

Modern HEP experiments produce an enormous data flow; a typical experiment writes several reels or cassettes of magnetic tape every hour. Although the situation will change with the next generation of experiments, where much larger computing power will be available on-line, today the full analysis of such volumes of data cannot realistically be performed on-line; data resulting from experiments written onto magnetic tape are analyzed on large mainframes either on site or at the collaborating Laboratories. The final product of an HEP experiment being in fact a publication of the results, the data resulting from analysis are very often presented in graphical form, in particular in the form of histograms. Therefore, it is essential to make available to CERN users sophisticated and powerful graphics features; it is also highly desirable to link such a facility to desktop publishing systems. For this important application many different programs have been developed by CERN and by the collaborating researchers during the last twenty years. The package used as a case study in this paper is probably one of the most advanced in the area of data analysis and data presentation.

3. Software development

During the last twenty years, CERN has played a leading role as the focus for development of packages and software libraries to solve problems related to HEP. The results of the integration of resources from many different Laboratories can be expressed in several million lines of code written at CERN during this period of time, used at CERN and distributed to collaborating Laboratories. Nowadays, this role of software developer and distributor is considered very important by the entire HEP community. In the process of developing software to be used in many different places, several key goals have to be met and a number of criteria and rules are to be observed. We list in the following those which are felt to be the most important ones:

3.1 Portability

Computers used at CERN and in the collaborating Laboratories are all but homogeneous. In practice, it is neither possible nor desirable to impose standards as concerns the hardware to be used at CERN and in all collaborating Laboratories. Many different manufacturers and product lines are involved; just to quote some of the most important ones involved, we could mention IBM, DEC, Apollo and Cray (one mainframe has recently been installed at CERN). It is important to observe here that the average physicist is not, and should not be, a professional programmer. This explains in some way the relative inertia of our colleagues in accepting new methods of approaching computing. As an example, punched cards were finally phased out on the CERN site only at the end of 1986, when all CERN users were converted to terminal access to computers in a time-sharing environment. Nowadays, we are just starting the next phase, consisting of the elimination of dumb terminals and their replacement with powerful workstations, backed up by a few very large machines, still needed for the provision of large CP power for batch processing, and of large backing storage. Needless to say, this evolution process will take several years to be completed, at least for budgetary reasons! In this kind of environment, an analysis and/or data presentation package must be able to run equally well:

- in batch processing on the mainframe,
- interactively, via dumb terminals in time-sharing on the mainframe,
- and interactively on the workstations.

Therefore portability is the password for the successful implementor of software packages. There are several aspects of portability that should be taken into account: portability of the source code, the support of the code, the interface to the operating system, the interface to the local data structure and storage, etc. This general need led CERN quite a long time ago to develop also portable software tools, e.g.: source code management systems, data structures management systems, mathematical and utility libraries, etc., which are either completely portable or machine-independent.

3.2 Modularity

Very often a package is not the opus of a single person, but many different developers collaborate in the writing of different modules of the package. It is also common that these implementors come from different Laboratories in different Member States. Therefore, packages must be developed in a modular way, in order to permit orderly independent development of different modules by different authors. This raises problems of standard coding rules and practices, and interface standards between modules. For a long time CERN has been trying to adopt and to enforce, whenever possible, rules to obtain code uniformity in software developed both locally and at the collaborating Laboratories.

3.3 Backward compatibility

This is one of the most difficult goals to achieve. In fact, once a package has been released for production, there is the tendency, or better, the systematic habit of many users to build around it their own specialized systems. The withdrawal of even minor features in an improved version of the system will inevitably provoke strong negative reaction from such users. In spite of this handicap, development of new, more modern and powerful packages cannot and should not be stopped by taking the backward compatibility as a main rule and sometimes backward compatibility has to be simply forgotten if it cannot match the desired improvements.

3.4 Adoption of standards

Standards are constraints that ensure the evolution — not revolution — of computing. (Gordon Bell)

When a large number of computers of different size and brand are used on the same site, it is important to adopt standards on those computers. Probably the only area where standards were very quickly adopted within the HEP community was that of a common programming language. The main reason was just historical: at the time when computing entered HEP applications no viable programming language other than FORTRAN was available [3]. Today, FORTRAN is virtually the only programming language used by physicists at CERN, and it is in fact used by them as in the past the slide rule was used by the engineer. For the reasons explained in the following, it took much more time to adopt international standards in other fields, e.g. computer graphics.

4. Computer graphics at CERN

It was recognized very early on that the provision of graphics facilities could be an important factor in helping the physicist in the analysis of experimental data and in the production of the results in graphical form. Development in the area of computer graphics started very early; operational systems were available to the physics community already in the late 1960s [4], [5]. These systems used pen plotters and the first models of calligraphic CRT displays. Several hundreds of graphics devices are operated today on the site. They range from simple DVST terminals, raster-scan Tektronix-compatible terminals, to powerful workstations (Apollo, DEC, Megatek, etc.). Hard-copy facilities are provided by electrostatic plotters and laser printers, installed both at the Computer Centre and remotely. Remote pen plotters are also used for specialized applications (CAD). In addition, hundreds of personal computers are exploited on the site for many different applications. The development of computer graphics software started very early at CERN (in 1964). At the time, we were just in the infancy of computer graphics, and no real standard package was available or even thought of. Therefore CERN took the initiative and developed its own device-independent graphics package, called GD-3 [6]. It is surprising to note that rather modern concepts were already present in the original implementation of GD-3, e.g. the concepts of device-independence and of the metafile. The system has been implemented in various versions on all computers in use at CERN, and also installed in more than 100 external Laboratories collaborating with CERN. The system proved to be extremely successful, and only today is it replaced with the present graphics standard, GKS [7] and [8]. A comprehensive guide to graphics software available at CERN is given in [9].

5. The case study: PAW

PAW, [10] and [11], is essentially an interactive system which includes many different software tools, strongly oriented towards the data analysis and data presentation. The system integrates several software tools, some of them already existing. PAW is conceived as an instrument to assist physicists in the analysis and presentation of their data. It provides interactive graphical presentation and statistical or mathematical analysis, working on objects familiar to physicists like histograms, event files (n-tuples), vectors etc. PAW is intended to be useful to any physicist with minimal notions of computing, at least at the basic level of usage. It requires, in principle, no long learning phase before getting started: PAW software is meant to guide a user through the possibilities he has, if desired in a fairly verbose form and using modern user interfaces (menus). Simultaneously, it is intended to become a powerful and efficient tool in the hands of a more experienced user, with practically limitless extension possibilities through added programs at the FORTRAN level.

As for other well-designed interactive systems, a number of rules were obeyed in the design phase: those which were felt to be the most important are listed in the following:

1. **Appropriate internal organization:** the user should be able to work on basic entities, which could be either single numbers, vectors, n-tuples, histograms, etc., using basic operators working on the complete entity, without any need to specify directly in the command that the entity is not a single number. In this respect, PAW could be considered as a superset of normal programming languages (e.g. FORTRAN which could be defined as a "single number" computing tool) or, better, as a "object-oriented" system.
2. **Rapid response:** we consider rapid response imperative for this kind of system; the response time for simple operations should be of the same order of magnitude as the human reaction time. For more complicated operations the response time may be longer but the user, on request, should be kept informed about the progress of his problem solution. Also, but depending upon the hardware in use, when displaying graphical information, the response time has to be very good; of the order of a second for a simple graph (on workstations!).

3. **Direct user control:** the user must have full control over what the computer is doing. He must be able to see at any moment intermediate results and to follow step-by-step the execution of his algorithm. He should be able to stop the process with no loss of data in emergency cases (e.g. when his procedure is running in an endless loop). Facilities must be available to the user that enable him to request and control hard copy of data and graphs.
4. **User friendliness and system forgiving of users' errors:** the syntax of the language should be particularly simple, and the system itself error-forgiving in the sense that errors must be detected immediately upon the entry of the command, and never fatal. Error messages should be absolutely clear and use a terminology which is familiar to the user. A powerful on-line help facility ought to be planned from the outset. Awkwardness of operating systems and graphic packages, whenever possible, should be hidden behind an appropriate human-engineered interface. On the other hand, it is important to permit the access to standard system functions, like the editor, at any time, in a completely transparent way, with no need to leave the package to access the system function.
5. **Simple on-line programming procedure:** the user should be able to define procedures (macros) and to store them under some name. He should be able to call on such a predefined procedure at some later time by means of its associated name. Editing and manipulation facilities are also required, possibly using the same software tools standard as the system where the package is operating. This facility permits the easy development of individual sub-systems.
6. **Availability of data:** of vital importance is the possibility of making data available to the system whether they reside on the same computer system or are stored remotely. Facilities to access data on heterogeneous networks are also very important.
7. **Flexibility regarding user needs:** the success of the system will largely depend upon an appropriate choice of basic operators. The choice of built-in operators should be based on previous experience, but current user needs should dictate the design as well.

The system has to be considered completely open-ended, in the sense that the user is able not only to personalize his own version, for instance using the mechanism of the KUIP macros (see section 5.2), but he is also able to create his own interactive language adapted to his particular application, by using the command definition facility which is the basis of KUIP.

In order to quantify in some way the complexity of the system we could just quote that the total number of lines of code of the seven modules composing PAW is well over 100,000. This figure does not include the underlying graphics package, the mathematical routines or the data structures management system.

Figure 1 outlines the various components of PAW and their mutual relationships.

In what follows we will examine some particular characteristics of the basic components of PAW.

Figure 1: Structure of the PAW system.

5.1 HIGZ

The package HIGZ (High level Interface to Graphics and ZEBRA), [12], was originally implemented in order to provide the graphics interface of PAW. However, HIGZ has been designed in a way that it is possible to use it independently. Graphics packages like GKS define a standard interface between user programs and devices. The use of only one such package, GKS, results in portability of application programs between systems on which GKS is installed, and makes the application programs largely device-independent. These packages, however, have limitations. They do not provide the high level functions (axes, graph, etc.) necessary for a data presentation system. There are always (sometimes minor) differences in the actual implementations on different computers (the so called "installation-dependent-parameters"). They do not foresee an acceptable way of recording large volumes of graphical information in compact form with a convenient access method for later manipulation. The GKS metafile is conceived as a vehicle to communicate series of pictures between computers, but not for their later manipulation. Also, the acceptance of GKS, in particular by Laboratories outside Europe, is modest, and thus it is not a standard that the High Energy Physics community can restrict itself to exclusively.

The following requirements were considered essential for the graphical output of PAW:

- The PAW picture data base must be fully transportable, as compact as possible, and accessible in direct access mode.
- It must have easily accessible graphical units (pictures) for later manipulation.
- The picture data base must be completely independent from the underlying basic graphics package and, a fortiori, from different implementations of the same graphics package.

These requirements are not restricted to PAW; they are common to many applications existing or under development. HIGZ is an interface package aiming at graphics applications of any nature, provided the level of functionality is similar. The package is basically a **thin layer between the user program and an underlying graphics package**, offering the following advantages:

- An interface to a standard data structures management system (ZEBRA), and through it a mechanism to store graphics data in a way which makes their organization and subsequent editing possible and easy. The picture data base is highly condensed and fully transportable.
- A picture editor is part of the package. It allows merging of pictures, editing of basic graphics primitives, operations onto HIGZ structures, etc.
- A GKS-like user interface to the graphics package, keeping the program independent of the graphics package installed.

The level of HIGZ was deliberately chosen to be close to GKS and as basic as possible. This makes the interface to GKS a very simple one, and preserves full compatibility with the most important underlying graphics packages. HIGZ does not introduce new basic graphics features, and does not duplicate GKS functions. On the other hand, some graphics macro-primitives are implemented, providing very frequently used functions, such as histograms, full graphs, circles, bars and pie charts, axes, etc.. HIGZ is presently interfaced to several versions of GKS, it being possible to select the chosen version of GKS at compilation time. The system is articulated into four main sets of functions:

- Basic graphics functions, interfacing to the underlying graphics package, with calling sequences identical to those of GKS.
- Higher-level macro-primitives, and the related control routines.
- Data structures management functions, interfacing to the data structures management system (ZEBRA).
- Picture editing functions.

The user is able to invoke any of these sets of functions, simultaneously or not. This is particularly useful during an interactive session, as the user is able to "replay" and edit pictures previously created, with no need to recall the application program, but just accessing the picture data base.

5.2 KUIP

KUIP (Kit for an User Interface Package) [13] is an interface program for any application based on interactive input of commands. It takes care of command input and of the parsing of commands; it verifies their syntactical correctness, and then invokes the appropriate application routines. In fact, KUIP could be defined as a syntax-directed compilative interpreter, as the basis of it is the so-called command definition file, which constitutes a complete description of the command language. Although KUIP and the associated definitions and tools have been developed in the context of PAW, KUIP has to be considered a quite general tool. We consider it very important to offer different interaction modes, adapted not only to the different hardware in use, but also to the different levels of user's experience. KUIP has basically several modes of interaction with the user:

- Call mode: the command line is given as the argument of a KUIP routine, which executes the command as if it were entered interactively by the user.
- User-Command mode: the commands and parameters are typed to the terminal by the user, in response to the KUIP prompt.
- Macro-Command mode: the command lines are read from an editable command file (called macro) and executed.
- Alpha mode: the commands are chosen by traversing (i.e. scanning through the branches) the command tree displayed on the terminal and the parameters are typed in.
- GKS-Graphics mode: the menu presentation is done using GKS. The commands with their associated parameters are chosen by traversing the command tree displayed by the graphics package GKS. (A device with a mouse and multiple windows is recommended.)
- Bitmap-Graphics mode: the menu presentation is done using high-level graphics packages on bitmap workstations. A machine independent implementation using the industry standard X/Windows is being studied.

An important feature of KUIP is the fact that style switching is permitted at any time: when in command mode the user can switch to one particular menu mode by entering the appropriate command. In the various menu modes, commands are composed by selecting elements from proposed sub-menus at different levels. When the lowest level is reached the parameters, if any, are prompted for, then the selected command is executed. Commands or command elements have some information associated which will be used by KUIP in order to minimize the remaining application-dependent dialogue. This information consists of parameters associated to commands, and of explanatory text called guidance to help the user to recall the usage of the command itself.

To give the possibility of reinspection or repeated execution of sequences of commands, KUIP keeps track of the last command lines entered, which can be recorded in a history file. Possibilities exist to manipulate and re-execute such a command file. In order to avoid verbosity, often useless and annoying for the experienced user, abbreviations are possible which may still retain partly the original mnemonic value. A rather interesting feature provided by KUIP, which to our knowledge is also unique, is the possibility to generate automatically the user documentation in text processing format, ready to produce the user manual, just by invoking the text processor chosen by the user (e.g. SGML, TEX, etc.). One of the by-products of this facility is to obtain documentation which is always up-to-date.

5.3 HBOOK/HPLOT and the basic objects of PAW.

As mentioned above, data resulting from HEP experiments are often presented in the form of histograms. The facility for producing this particular kind of graph is so important in HEP that our physicist colleagues have since a long time altered the semantics of the word "histogram". An "histogram" in HEP slang is not only "A representation of a frequency distribution by means of rectangles whose widths represent class intervals and whose areas are proportional to the corresponding frequencies"¹, but also the complex of numeric and non-numeric information, enabling the data presentation package to build-up the final histogram. In the following, the word "histogram" will be used in the HEP slang sense.

HBOOK, [14] and [15] is a FORTRAN-callable histogramming facility, whose purpose is to define, fill and edit histograms, scatter plots and tables. The main application of HBOOK is to summarize basic data derived from experiments or from the subsequent analysis process. It can also be used to represent real functions of 1 or 2 variables. Essentially for historical reasons, the basic output is formatted for the printer, but a graphics interface is provided by the HPLOT package, [16]. HPLOT is a FORTRAN-callable facility for producing HBOOK output on graphics devices. Its main design objective is to be able to produce drawings and slides of a quality suitable for talks and publications. An interactive version of HBOOK and HPLOT called HTV, [17] has existed since 1981. The functionality of both HBOOK and HPLOT has now been incorporated into PAW in a fully interactive mode.

The basic object types on which PAW is able to operate are the following:

- HBOOK objects which may have been created in a batch environment and kept on an external data store, i.e. disk or tape. Such objects are, of course, also generated inside PAW. They are three main categories:

Binned data in 1 or 2 dimensions ("histogram"),

Event data with a fixed number of entries per event ("n-tuple"),

Statistically accumulated data in 2 dimensions ("profile graph").

Operations that allow transformations of these data into each other (where possible) exist in PAW: a 1-dimensional histogram can be produced from an n-tuple or from a 2-dimensional histogram ("projection"). 2-dimensional histograms or profile plots can be generated from an n-tuple. The internal format of HBOOK objects is that of the HBOOK support package ZEBRA [21], but the only aspect a PAW user should know about is the naming convention which is used for these objects, and the rules that govern transfer from (volatile) memory to (permanent) disk, deletion or overwriting, etc.

- Vectors: vectors of real or integer elements. Vectors may be created in PAW, loaded from HBOOK objects, typed in by hand, read from files generated by FORTRAN programs, and, of course, manipulated by operations.
- Functions: they may be written externally or inside PAW, using the COMIS [20] FORTRAN interpreter and following the FORTRAN77 syntax. PAW functions may refer to other functions (interpreted or compiled) and to PAW vectors.

¹ Webster's Third New International Dictionary of the English Language, Merriam Webster, 1981.

- Pictures: all graphics operations in PAW, from the most basic to the complete plot, may be recorded both as metafiles for shipping to various output stations, and as PAW pictures, which can be brought to other PAW systems for display or graphical editing.
- Macros: as PAW can operate in a command mode, executable commands can be collected into a file for repeated execution, possibly under control of parameters. Macros may also be created and edited using the local editor of the host computer.

Figure 2: Examples of HPLLOT output.

5.4 SIGMA

SIGMA (System for Interactive Graphical Mathematical Applications) [18], [19] is a programming language for scientific computing whose major characteristics are the following:

- The basic data units are scalars, one-dimensional arrays, and multi-dimensional rectangular arrays; SIGMA provides automatic handling of these arrays.
- The calculation operators of SIGMA closely resemble the operations of numerical mathematics; procedural operators are often analogous to those of FORTRAN.
- The system is designed to be used in fully interactive mode; it provides convenient facilities for graphical display of arrays in the form of (sets of) curves. In fact, in order to avoid duplications, the original graphics commands of SIGMA have been replaced by those provided by PAW.

- The user can construct his own programs within the system and has also access to a program library; he can store and retrieve his data and programs.

SIGMA, which has to be considered as a system for interactive on-line numerical analysis problem-solving, has been designed essentially for mathematicians and theoretical physicists. It has been operational at CERN for several years on CDC CYBER computers, and its main features have been now incorporated into PAW.

SIGMA commands are executed in the framework of a special language processor, hence outside the PAW command processor. They can also be assembled into PAW macros. SIGMA-generated arrays are stored as PAW vectors and therefore are accessible to PAW commands, and PAW vectors are accessible to SIGMA.

Figure 3: Some mathematical functions handled by SIGMA.

5.5 COMIS

COMIS [20] is a FORTRAN interpreter, allowing the user to write and execute in interpretive mode FORTRAN subprograms. This facility is of great importance, as in this way the user has the possibility to write his own data analysis procedures, for instance his own selection criteria, minimization functions, etc..

5.6 ZEBRA

ZEBRA [21] is the CERN standard data structures management system. It is a totally machine-independent package allowing the FORTRAN user to create dynamically data structures at execution-time, to manipulate such structures, and to transport them between different computers, using external storage media. ZEBRA provides a significant extension of the power of FORTRAN, in general at an insignificant cost in terms of execution-time overheads. However, even that small cost is tiny compared with the extra time which would otherwise be wasted in developing large programs using only the conventional facilities. One of its most important features is also to provide a convenient user interface for devices allowing direct access. In the context of PAW, this facility is essential for the manipulation of pictures.

5.7 Graphics features of PAW.

As mentioned above, the final scope of PAW is to produce publication-quality graphs, and it is therefore essential to make available the most sophisticated and powerful graphics features. It is also important to have these facilities linked to desktop publishing systems. A large spectrum of graphics options is available at the user's fingertips: histograms, scatter plots, contour plots, error bars, bar charts, column charts, pie charts, surface and "lego" plots, etc. In most cases, and certainly for the inexperienced or casual user, graphs can be produced by just providing a minimum of information, e.g. where the data are stored and what kind of graph is desired, and then the system produces automatically the graph, using default parameters and with no need for further programming effort. However, the more experienced user is seldom satisfied by graphs produced automatically, and therefore many facilities are provided, permitting the user to alter the default parameters related to the graph, enabling him to produce a fully personalized picture.

6. DOs and DON'Ts

There is no pretention here to give the universal recipe for the writing of successful software packages, but just to relate what has been our experience in this area.

6.1 DO

We list here some good practices which are always worth following:

1. Carefully plan a good human interface: with almost no exception, current operating systems present a human interface which seems designed for masochistic users. Efforts spent in designing and implementing a good human interface for the package will be always amply rewarded.
2. Design for maintenance: it is commonly recognized that maintenance of software products is often more expensive in terms of time and manpower than the actual development. It is always worthwhile to invest effort during the design phase in order to plan for facilities to ease the future maintenance of the system.

3. Need for a prototype: very often users' needs and wishes do not coincide with those of the system designers. Users' feedback is invaluable during the design and implementation phases. The most effective way to obtain such a feedback is to make available to a few selected users a prototype, even incomplete, of the system, with the clearly declared purpose to permit experimentation and obtain feedback. The designer must be prepared to discard completely the prototype and to start afresh the programming of the future production version of the system.
4. Be prepared to revise basic design: we want here to quote an example: the basic syntax of PAW was originally designed on a "verb/predicate" structure. During the development, we quickly reckoned that this structure was orthogonal to a good human interface when in a menu-mode. Therefore we decide to reverse the syntactical definition to a "predicate/verb" structure. Thanks to the command-definition-file facility of KUIP, the total re-definition took a total of four working hours.
5. Try to resist the moving target syndrome: one of the most common mistakes is that of changing specifications of the product before having produced and tested at least a prototype. This very often results in loss of effort and time.
6. Adopt standards: in particular, in the CERN environment:
 - a. Write code in "pure" FORTRAN 77.
 - b. Use standard graphics packages: GKS in the CERN case.
 - c. Use standard data structures management: ZEBRA in the CERN case.
7. Use standard source code management systems: there two main reasons for which the use of source code management systems is strongly recommended: firstly they permit easy maintenance of the system and, more important, parts of the system which cannot be written in a machine-independent way can be easily selected at compilation time. Needless to say, the source code management systems ought to be portable. An example of these systems, widely used in the HEP area, is PATCHY [22]. Unfortunately, the human interface of these systems is rather poor; development is under way in order to improve this aspect, (e.g. CMZ, [23]).
8. Use debugging and optimization tools: most manufacturers offer nowadays a number of debugging tools which are invaluable for the efficient and rapid testing of a software package. We just want to quote here the VMS symbolic debugger [24] and the Apollo DEBUG feature, as examples of very efficient and powerful debugging packages, although their user friendliness is rather poor. Another tool available on Apollo worth mentioning here is DPAT (DOMAIN Performance Analysis Tool) [25] invaluable to optimize programs. One of its main features is the ability to work on binary modules, with no need for special steps to be taken at compilation time. The output of DPAT is a precise indication of the time spent by the machine (in percentage) in executing the different subprograms, while it is possible to obtain the same output in real time in graphical form. Other possibilities offered by DPAT are to "replay" the session and to show the calling tree, also in graphical form.

7. Conclusions

We have discussed in some detail the development process of a large software package making use of computer graphics in the environment of a large scientific research Laboratory. The present situation at CERN was presented, and our experience with it during the last years was related. The development work done at CERN in connection with these systems was outlined.

8. Acknowledgments

The author wishes to thank Rene Brun, who has the direct responsibility for the development of many application packages at CERN, and Pietro Zancarini, main implementor of the KUIP system, for their very important comments and contributions to the writing of this paper. We also recognize the role of R. Bock in the original concept of PAW.

Particular thanks go to Jackie Turner, for invaluable help in the final preparation of the paper.

9. References

1. P. Zanella, *Computing in High-Energy Physics: Facing a New Generation of Experiments*, Proceedings, Information Processing 83, editor R.E.A. Mason. IFIP, North Holland Publishing Company, 1983, pp. 79-88.
2. M. Metcalf, *Computers in High Energy Physics*, Advances in Computers, 25, Academic Press, 1986, pp.277-334.
3. M. Metcalf, *Aspects of FORTRAN in large-scale programming*, CERN/DD/82-18, November, 1982.
4. P.M.Blackall, J.C.Lassalle, C.E. Vandoni and A.Yule, *High Energy Physics Applications in Computer Graphics-Techniques and Applications*, edited by R.D. Parslow (Plenum Press, London, 1969), pp. 149-160.
5. L.B. Smith, C.E. Vandoni, *Graphical Man-Machine Interactive Systems for Numerical Problems: PEG, a Special Purpose System and GAMMA, a General Purpose System*, CERN Report 70-23, August 1970.
6. R. Miller, *Program Library Long Write-up J510*, GD-3. CERN, Data Handling Division, May 1976.
7. *Graphical Kernel System (GKS) Functional Description*, ISO 7942 and ANSI X3.124-1985, ANSI, New York, 1985.
8. D.R.Myers, *GKS/GKS-3D Primer*, CERN/DD/US/110, August, 1987.
9. *Guide to computer graphics at CERN*, CERN/DD/US/111, August, 1987.
10. R.Bock, R.Brun, O.Couet, J.C.Marin, R.Nierhaus, L.Pape, N.Cremel-Somon, C.Vandoni, P.Zanarini, *PAW - Towards a Physics Analysis Workstation*, Computer Physics Communications 45 (1987), pp. 181-190.
11. R.Brun, O.Couet, N.Cremel-Somon, P.Zanarini, *PAW - Physics Analysis Workstation*, CERN Program Library Q121.
12. R.Bock, R.Brun, O.Couet, R.Nierhaus, N.Cremel-Somon, C.E.Vandoni, P.Zanarini, *HIGZ Users Guide*, CERN Program Library Q120.
13. R.Brun, P.Zanarini, *KUIP Users Guide*, CERN Program Library I202.
14. R. Brun and P. Palazzi, *Graphical presentation for data analysis in particle physics experiments: The HBOOK/HPLOT package*, EUROGRAPHICS 80, Proceedings of the International Conference and Exhibition, University of Geneva, Switzerland, 3-5 September 1980, editor Carlo E. Vandoni, North-Holland Publishing Company, Amsterdam, 1980, pp. 93-104.
15. R.Brun, D.Lienart, *HBOOK users guide*, CERN Program Library Y250
16. R.Brun, N.Cremel-Somon, *HPLOT users guide*, CERN Program Library Y251

17. R. Brun, *HTV, the Interactive Version of HBOOK+HPLOT*, CERN, DD/EE/80-5, DD/US/87, November 1981.
18. R. Hagedorn, J. Reinfelds, C.E. Vandoni and L. Van Hove, *SIGMA, A New Language for Interactive Array-oriented Computing*, CERN Report 73-5 (1973). Also CERN Report 78-12 (1978).
19. J.Reinfelds and C.E. Vandoni, *Sigma 76*, Proceedings, Information Processing 77, editor B. Gilchrist. IFIP, North Holland Publishing Company, 1977, pp. 963-978.
20. V.Berezhnoi, R.Brun, S.Nikitin, Y.Petrovykh, V.Sikolenko, *COMIS Users Guide*, CERN Program Library L210
21. R.Brun, M.Goossens, J.Zoll, *ZEBRA Users Guide*, CERN Program Library Q100.
22. H.J. Klein, J. Zoll, *PATCHY Reference Manual*, CERN Program Library, October 1983.
23. R.Brun, *CMZ, a Code Management system using ZEBRA*, Private communication, February 1988.
24. *VAX/VMS Debugger Reference Manual*, publication AA - Z411C - TE, Digital Equipment Corporation, April 1986.
25. *Analyzing Program Performance with DPAK*, publication 008906, Apollo Computer Inc., Chelmsford, Mass., June 1986.

Carlo E. Vandoni -- Data Handling Division -- CERN 1211 Geneva 23 SWITZERLAND