The Next Generation of the NSF Interactive Graphics System — a Preliminary Requirements Document

Dave Love
NSF Software Group
(d.love@daresbury)

Edition 0.1 of 1/2/91 — drafty

This document is not claimed to be either complete or correct: it may be considered at 'α-test'. Comments are solicited, especially about other areas that should be covered, preferably with an indication of what to say since lack of inspiration is the reason a few aren't yet covered. Reports of typos are welcome (as would be a better acronym than NIGS!).

Contents

Chapter 1

Background

1.1 Raison d' être

This document attempts to define requirements for a replacement for the current NSF interactive graphics system ('Nigs'). For information on the significance of a requirements document in the software life-cycle, consult a book on software engineering e.g., [1, 2]. This document doesn't follow the usual scheme of such things since it is not written by the customer but rather *for* the customers to elicit comments on perceived omissions or mis-features and to make concrete the plans for what will be provided otherwise. (It was not expected that the 'customers' would produce such a statement of requirements spontaneously!) One wants to overcome conservatism to some extent, so as to allow progress in line with modern software, hardware and user-interface technology of which users may not be aware.

This requirements document is unlikely to be followed by the usual detailed specification document, although some aspects may be specified in detail. User interfaces like this seem best developed with a prototyping approach.

1.2 Interpreting this document

The language used here is not necessarily consistent. Words such as 'should' are usually meant to be imperative. Implementation restrictions might modify some of the requirements given.

1.3 The function of Nigs

The NSF interactive graphics system provides the user interface to acquisition and analysis facilities for experimental data acquired at NSF or elsewhere and analysed at any of the UK NSC-supported sites (Daresbury and several universities). It is a general facility; this document is not aimed explicitly at the EuroGam project, for instance, although EuroGam experimental groups are expected to be major users of the future Nigs. We aim to cover all the currently-used—not necessarily all currently-supported!—facilities and to attempt to improve, generalise and add to them to cover a wide range of experimental work.

1.4 Historical perspective

1.4.1 Concepts

The current NIGS system (modulo various upgrades) has been in use for a decade. It draws on previous experience and facilities from the long-dead Sigma3 acquisition/analysis system at Liverpool and Nero system at Manchester. Its vintage means that a serious re-fit should be considered in the light of progress in various areas over the last 15 years, although it was well ahead of its time[1]; the job of specifying an improved system is made more difficult by the fact that superior systems do not seem to have emerged in the nuclear structure world (an early pioneer of such techniques) during the life of the current system. (One potentially-interesting system, PAW [3] seems largely to provide examples of features to avoid!) However, lessons have been learnt from the use and development of the system. The most important one is probably that the previous design was pretty well right in the prevailing technological climate and it is in that light that you are asked to view current proposals from a similar source where they may be contentious.

1.4.2 Hardware platform

A more pressing need for re-development than age, though, which provides the opportunity for new facilities and approaches is the obsolescence of the hardware around which the current system was designed. This is already manifest in the forced replacement of the Sension displays by emulations using workstations—not an obvious step forward. A more traumatic change will come soon [4] with the need to replace the GEC 4000 series 'super-mini' computers with more powerful and cost-effective hardware in the form of *graphics workstations* (not to be confused with the old concept of the 4000s providing a multi-user 'workstation'). The rate at which computer hardware is now evolving makes forward planning tricky and means that one has to be prepared for a hardware base which changes with a period of, say, two or three years rather than the decade or so that we have been used to.

Hardware and software constraints on the system are outlined in Appendix A.

1.4.3 Recent work

There has been some recent prototyping work at Daresbury directed towards the system discussed here which has provided some ideas relevant to the requirements aspect of the system and to the subsequent implementation.

---

[1] Some of the concepts used which have only recently become popular are:

- direct manipulation techniques with light-pen(/mouse)-selected drop-down menus;
- an interpretative framework based on JCL rather than monolithic programs;
- the facilities of the graphics metafile system.

Chapter 2

Purpose

The overall purpose of NIGS is to provide fast, intuitive, extensible, integrated, interactive facilities for

1. *Visualisation* of experimental and related theoretical data of types discussed in chapter 4;
2. General *manipulation* of such data (see chapter 6);
3. *Presentation graphics* derived from the above functions (see chapter 8);
4. A graphical interface for *control* and *monitoring* of (possibly remote) systems for:
    - data acquisition;
    - setup of experimental equipment;
    - sorting;
    - verification of the systems' operation.

There is an intentional distinction between items 1 and 3. The expectation is that graphics for the former will be simpler, quicker and easier to use than the latter. *Integrated* implies that the various facilities are coordinated and can communicate with each other.

The system should co-exist happily with other tools in a general-purpose computing environment i.e., one should be able to run applications—like an editor or a mail tool—other than data analysis at the same time on the same display.

2.1 Restrictions

- Display of data will be restricted to x v. y or z v. (x, y), the latter represented as contours or images rather than 'wireframes', solid surfaces etc;
- Animation is not required;
- Other than red-green colour blindness, possible disabilities of users e.g., deafness, will not be taken into account;
- Foreign languages will not be supported (see §3.4).

Chapter 3

System structure

3.1 Generalities

The system is expected to be based on a *WIMP*-orientated networked *windowing system* with *multi-tasking* at the user level. This will fulfill the requirement that the analysis system is well integrated with the general computing environment and will provide the means for the most important departures of the new system from the old one. The primary means of interaction will be by *direct manipulation* (with a mouse), but it should be possible also to use it in a *command-driven* manner (by typing, commands as possible at present).

Consideration should be given to providing other such as

- knobs (as on the accelerator control system) to make interaction with analogue devices easier when controlling apparatus;
- the button box on the Sension system to provide customised short cuts, although keyboard function keypads may suffice;
- dials for changing range, as provided on the Liverpool Sigma3 system.

As well as direct manipulation, it should also be possible to *program* the system and it is important that it be readily *extensible*, even by the relatively inexpert, so that specialised or time-saving facilities can be incorporated according to the user's task in hand. In particular, there should be a well-defined interface to the system so that application code may be readily imported and adapted to use the facilities of the graphical interface, data base etc. The requirement to incorporate new application code thus (as well as experience and prevailing wisdom) means that the existing structure of a collection of relatively small programs glued together with an interpretive *shell* (currently GEC `COMM`) should be maintained. ('Shell' does not necessarily mean a `sh` derivative.) Such a feature is important to provide customisability and maintainability.

3.2 Shell facilities

The shell must provide an interactive general (i.e., Turing-equivalent) programming facility, allowing control constructs like **if**. . . **then**. . . **else** and **while**. It should also support internal variables of convenient types and thus means of assigning and extracting their values. Its primary purpose is *process control* by running programs (using operating system facilities) and passing instructions to parts of the interface. For running programs it will be helpful if it supports an interface for passing parameters along the lines of GEC `COMM`'s *proforma* system rather than the ad hoc parameter decoding of Un*x. The shell language must be interpreted, although it may be helpful to allow compilation for performance-critical applications. The choice seems to be between a macro-processor like `COMM` or `sh` and an evaluating system like Lisp. It is important that user-defined routines/macros/functions (according to the implementation) are available. Timing facilities are important for experiment control; the ability to achieve the same effect as `COMM`'s `EVENT` facility is needed.

3.3 Programmability/extensibility

There are essentially two kinds of programming/customisation/extensibility envisaged. *programmability* involves relatively trivial use of existing facilities for purposes such as automating tasks like looping through a set of spectra doing the same operation on each. *Extensibility* means adding significant facilities like importing a program that writes a paper about superdeformation given a γ-γ dataset as input and requires a graphical interface (presumably only for minor editing of the output!).

Repetitive tasks can be accommodated either by allowing users to write macros in an appropriate *scripting language* (c.f. *shell scripts*) or by *learning* a set of actions consisting of direct manipulation with the mouse and/or typing of relevant input. EMACS provides a model for both facilities (and several other facilities discussed in this document). Both ways of handling such tasks should be available.

Graphical extensibility should be provided following the HyperCard model of persistent, directly-manipulable interface components. Thus one should be able to copy a 'widget' from an existing application, paste it into another one and then customise its behaviour with a suitable scripting language.

3.4 Modes

It may be desirable to provide one or more *modes* e.g., for expert or naïve userss to determine trades off of perspicuity against speed and power, for instance. The possibility of providing foreign languages modes (as, for instance with alternative 'resource files' in MacApp and GEM, `.Xdefaults` with X11 or NeWS dictionaries) is probably not worth the effort[1]; (online dictionaries/phrasebooks may be obtainable!).

---

[1] Parameterising messages, help text etc. will tend to compromise its quality in the author's opinion and fighting this tendency will take the effort. The existing NeWS prototype system does implement a polyglot button/menu system, however.

3.5 Interfaces to other systems

3.5.1 Data-acquisition/sorting system

The main requirement will be to issue commands to the system for tasks such as

- stop/start;
- change configuration (e.g., sort commands);
- change tape/file, premount tape;
- save datasets to disc.

Display of parameters such as tape status will also be required. The display of sorted/histogrammed data from such systems is automatically included in the required uniform data base interface(§4.2.1). Data base window objects should be freely exchangeable between the sorting and analysis systems (preferably by some sort of cut and paste) to allow sort windows to be edited graphically in comparison with a spectrum of the relevant parameter(s).

3.5.2 Apparatus control and monitoring

The main need will be for graphical devices for readout and control of analogue parameters e.g., dials, knobs, barcharts. A physical knob would probably be useful. It is expected that *mimic diagrams* of the system to be controlled will be available as currently provided in the NSF accelerator control system. Such a diagram is a schematic of the apparatus with active components of the diagram corresponding to apparatus components which may be selected for control or monitoring.

3.5.3 Exception server

See §9.3.2.

3.6 Slice server

There will be a need for a separate server to deal with operations on large datasets. The criterion for being 'large' is that they won't fit realistically in main memory in the graphics system and/or would take an unacceptably long time to load into it from mass storage (which will probably be accessed over the LAN). Typical tasks for the server would be making a projected slice of a dataset residing in its local storage for graphical display, the parameters being defined by a window edited using the display of a previously-obtained projection.

3.7 Application programs and libraries

Libraries (callable from Fortran77, Fortran90 (when available) and C) must be provided to support the operations defined in this document. These will include data base!operationsdata base operations, control of interface components (e.g., window display), message passing to other systems, command line argument decoding, graphics functions. These will define a clean interface against which application programs can be written or imported. The way this works will obviously depend on the implementation of the shell.

Chapter 4

Data base aspects

The system is required to deal both with data structures (*spectra*) obtained directly from experiments by *sorting* or *histogram readout* and with related data typically derived from analysis of experimental or data from theory (e.g., fits to the data). This chapter describes the data types required to be stored and operations the data base system by which they are accessed is required to support.

4.1 Data types

Several different types of item may occur in the data base, as follows.

4.1.1 Spectra

4.1.1.1 General definition

Spectra comprise *counts* associated with an array of *channels* which comprise uniformly-spaced bins. Spectra can only be displayed if they are *1-dimensional* ($x \mapsto counts$) or *2-dimensional spectra* ($(x, y) \mapsto counts$). Higher dimensionality may be required, but spectra with three or more dimensions will not be capable of being displayed. x and y are signed 16-bit integer quantities. *counts* are signed 16- or 32-bit integer quantities, chosen according to the maximum intensity of spectra. The range in (x, y) which may be displayed will be limited. To visualise them it will be necessary to make lower-dimension *sections* (*cuts* or *slices*) which can be displayed, either directly or following storage.

4.1.1.2 Stored and live spectra

Spectra occur in two types:

Stored spectra reside in permanent storage (presumably disc), having been created by sorting, manipulation of other spectra or *saving* live spectra;

Live spectra are resident in the data-acquisition/sorting system rather than in disc files in the usual user filespace. They are potentially continually updated, certainly updateable, by the acquisition/sorting system.

4.1.1.3 Associated entities

Associated with spectra there may be the following optional entities:

A *title* giving a brief description of the spectrum. This should be a text record at least 80 bytes long. It is expected to be modified by operations which create new spectra from old ones to give e.g.,

```
Total projection, y-scaled (y'=0+.5y)
```

Comments giving a more detailed description of the spectrum. Comments are textual records with a size expected to be a paragraph or so describing, for instance, the conditions under which the spectrum was generated. (Not currently supported);

A *calibration* which maps between channel number and physical units (typically keV or MeV). Calibrations are allowed for any dimensionality of spectra supported by the system, but with independent functions for each dimension i.e., no correlations. As a minimum, the method of Wooff [5] should be employed for energy calibration in terms of polynomials up to cubic. To generate the calibration a set of sufficient points to fit is required, comprising channel (as a real number), corresponding energy and errors on both. The calibration is stored as the coefficients of the fitted polynomial along with the error matrix. It has associated *units*. Software should be aware of common conversion factors e.g., keV↔MeV. The implementation should not preclude adding other forms for the calibration function, should they be required in the future. One possibility is a square root function if spectra are re-binned to keep peaks roughly the same width. The coefficients of the polynomial may have associated errors which must propagate through to the physical unit derived from them. The ability to use more general function when appropriate should be available;

An *efficiency calibration* which specifies how to correct for detector efficiency, probably best expressed as an interpolation function. A weighted cubic spline is probably best. It will be specified in terms of a fit to points comprising (channel, efficiency, channel error, efficiency error) with automatically-chosen knots so as to satisfy some smoothness criterion which should be user-specifiable. The NAG library [6] provides suitable technology. Note, however, that, error estimates for the resulting calibration are not easily determined. The fitted calibration will be stored in terms of a variable number of knots, although there will be an upper limit to this number which the fit has to observe. This will be used by application programs which determine peak intensities, for instance. Multi-dimensional data are treated as for energy calibrations. (Not currently fully supported);

An *error spectrum* which defines the *standard error* associated with the counts in each channel for use by analysis programs such as least-squares fitting. The error spectrum may be given implicitly by specifying *counting errors* i.e., the standard error on the counts is derived assuming they obey Poisson statistics. Spectra containing negative counts cannot have counting errors. Errors are real quantities. (Not currently fully supported.);

Partitions whereby sections of a spectrum may be referred to by the partition name. Each name is associated with a range of channels. This facility is useful for dealing with *routed* spectra in particular. The current convention of referring to partitions as ⟨*spectrum*⟩[⟨*partition*⟩] should be retained. This will exclude [and] (possibly just [) from 'normal' names describing unpartitioned spectra. Partitions are currently referred to as *sub-ranges*.

4.1.1.4 Scalers

Scalers are effectively degenerate spectra with only one 32-bit channel, but need to be supported as a separate type. They may only have associated *comments* and *titles*.

4.1.2 Non-spectral data

Non-spectral data only occur in *stored* form, not *live*. They may have associated *titles* and *comments* as for spectra.

4.1.2.1 Graphs

Graphs comprise real (x, y) data which are not required to be uniformly binned, unlike spectra. The data points may have associated (real) *errors* in x and/or y. The 'plus' and 'minus' errors may be different. A better term to avoid confusion with the mathematical idea of a graph would be welcome. (Graphs are not currently supported.)

4.1.2.2 Maps

Maps comprise real (x, y, z) data where the x and y values are not required to be uniformly spaced, unlike 2-dimensional spectra. Errors on the z values may be given[1]. The 'plus' and 'minus' errors may be different. (Not currently supported.)

4.1.3 Notes

Notes contain text rather than 'data' in the sense usually understood by scientists. They provide some description of the data at the current level of the hierarchy as distinct from the comments in spectra etc. A typical use is to contain the sort commands used to generate a set of spectra. Notes are currently not supported explicitly but are available by using text files in an SDB catalogue.

4.1.4 Window records

Window records are used by the sort system [7] to define *gates* which filter the data flow in the sort. They need to be accessible by the graphics system as well as the sort system so that they can be displayed with spectra formed from the parameters that they filter and re-defined interactively to select the appropriate region of the spectrum. They are also used to define slices of multi-dimensional spectra. They have *titles* and *comments*. The 'definitive' version of a record presumably does not

---

[1] Is this sensible?

reside in the data base under consideration here but in the sort system. It will, however, be helpful to store such records permanently along with the data they refer to.

4.2 User's view and implementation implications

Users need to see a hierarchical structure to the data base. They will want, for instance, to group data according to *experiment* and *run* within that experiment. Each component (a hierarchy or an individual spectrum) is referred to by a name which should be able to be fairly explanatory i.e., not too short or character set-restrictive. Typical examples might be:

```
Break-up_run_Jan_92
telescope1_deltaE
```

An individual spectrum etc. is referenced by a concatenation of hierarchy names and the name of the item at the lowest level.

These required features suggest that individual spectra etc. should be stored as individual files in the Un*x filing system. Then the directory structure provides the hierarchy and legal file/path names may be sufficiently general. This approach is adopted in the current SDB system, albeit with the requirement of descriptive names rather relaxed; maintaining it has benefits such as:

- familiarity to users both of Un*x and the current system;
- convenience of use and implementation—many of the required features are already provided by the filing system, including such advanced ones as distribution over the network.

It is thus made a non-functional requirement that the Un*x filing system provides the basis of the database for *saved* data, with spectra/graphs/maps/notes stored one per file (although it might be convenient to allow several scalers in one file). Individual files in the database should have an *extension* which indicates the type of the data item (1-d or 2-d spectrum, graph etc.). This will help the implementation to present only the relevant items for selection for a specific operation e.g., the user should not be presented with a list containing 2-d spectra when required to select one to overlap (§7.5).

Although most users are expected to be familiar with the filing system, the data base interface for naïve users should not require such familiarity.

4.2.1 Live spectra

The data base and display should present a uniform view to the user, whether the data are live or saved. Thus for instance, it should not be necessary to operate in a special *online mode* to display and analyse live spectra. It is important from a user's point of view that the design addresses this point carefully.

Since saved spectra are expected to live in the filing system, to maintain uniformity *live* spectra have at least to appear to reside in the filing system also. It would be most satisfactory to have them actually accessible as a virtual filesystem, but this is apparently un-implementable. Thus utilities and libraries that access live spectra should take care to treat them so that they appear to be files, probably living in a flat directory. The assumption is made elsewhere in this document that this is how live spectra appear.

4.3 Data base operations

Various operations must be supported on the data items. Some of these are trivially present if the data base is based on the filing system. They should all be available through a library that can be linked with application programs. Where operations are allowed on sets of items the members of such sets should be definable by *pattern-matching* of the names. Simple regular expression matching may not be most appropriate e.g., it can't pick out a general numerical range like '15–23'. The user will optionally be asked for confirmation when requesting a destructive operation like a deletion or over-writing or one awkward to undo like a multiple re-calibration.

4.3.1 Atomic operations

These operations relate to entire data structures in the data base, which may be spectra, graphs, maps, notes or hierarchies. In some cases the operation is considered to be obvious with no further explanation given. (See also §4.5.4.)

4.3.1.1 Copying

There are two cases:

1. Copying at the same level (the same directory). In this case the user or application must supply a new name;

2. Copying to a different level. The copy's location must be given and the name is maintained.

4.3.1.2 Deletion

The name of the item to be deleted should optionally be subject to confirmation.

4.3.1.3 Creation

It must be possible to create new (empty) instances of item types. Usually this will be done by application programs, often automatically. It is likely that users will require to be able to select different *policies* for generating names automatically in situations like re-scaling data. A typical such policy is to increment a trailing number in the item's name such that it is unique at the current level of hierarchy or, if the name does not already end in a number append '`1`' or '`01`'. Alternatively a new name may be solicited from the user; this will probably always be done if it is a new sub-tree of the hierarchy that is being created.

4.3.1.4 Interrogation

The parameters of the item must be accessible, such as a spectrum's maximum and minimum (x, y, z) values, calibration, dimensionality etc. In the case of a node of the hierarchy it may be desirable to extract the names of its siblings, but this is the only such information it is sensible to require. This might be considered a non-atomic operation (§4.3.2).

4.3.2 Non-atomic operations

These operations are intended to read or write *part* of the data structure. Again, some are thought to be self-explanatory.

4.3.2.1 General

Write title ;

Read title ;

Write comment adds a comment to the item;

Read comments Since multiple comments are allowed in an item, this operation should return all of them as a list;

Delete comment deletes a comment specified by its position in the list as returned by *read comments*.

4.3.2.2 Spectra

In the case of partitions the channel specifications for the operations below are relative to the partition origin.

Read channel contents ;

Write channel contents ;

Read channel block returns an array of channel contents rather than a single content (for convenience and efficiency). It requires a starting channel, the number of channels to read and an array to fill. The earlier dimensions of multi-dimensional spectra vary fastest, as per the Fortran convention;

Write channel block as per *read channel block*;

Read channel error ;

Write channel error ;

Read error block similar to *read channel block*;

Write error block ;

Read calibration [The format of a calibration needs defining];

Write calibration ;

Read efficiency calibration ;

Write efficiency calibration ;

Copy calibration Copies the calibration from one spectrum to another of the same dimensionality or to all those of this dimensionality beneath a specified level in the hierarchy;

Copy efficiency calibration Similar to *copy calibration.* ;

Actual energy or efficiency calibration of individual channel contents will be provided by the application library from the extracted calibration.

4.3.2.3 Graphs and maps

Since we expect graphs and maps to have relatively few data points operations are only supported to read/write the entire dataset.

Read graph data returns the number of data points and errors, given arrays to fill with them;

Write graph data write data and errors, specifying the number of points and type of error (x, y or x/y);

Read map data as per graphs;

Write map data as per graphs.

4.3.2.4 Window records

Records are expected to be accessed whole, rather than at the level of individual *gates*.

Read window ;

Write window .

4.4 Data integrity

Since the system is expected to operate on top of a distributed filing system it is possible for items like spectra to be modified (even deleted) underneath an application program that is trying to deal with them. It doesn't seem possible to prevent this in a sensible way. Some means of dealing with files that have changed this way has to be provided, however, if they are going to be updated by an application. At least the approach taken by EMACS should be adopted. This remembers the modification time on files that it has read in and warns, for instance, if a file onto which it is about to write back has been modified since it was read. Solutions involving locking seem too vulnerable to the lock not being undone, and, anyway one is hoping to be able to use standard system utilities on the files comprising the data base and so cannot prevent them being altered at will by anyone with write access.

4.5 External representation

An *external representation* (ER) of internal data structures is required that will allow easy portability of data to sites with different systems. (The *internal representation* (IR) is expected itself to be conveniently and efficiently transportable (e.g., by NIFTP) between NSF-supported sites, especially if it is just based on the filing system.) The requirements for the ER are outlined in the following. The current ER is described in [8].

4.5.1 Scope

The ER should support all the features of the data base such that transforming from IR to ER and then back to IR has no effect. Thus, in particular, the ER should be able to express the hierarchy of the IR. This is not currently done [8]. It is not clear to what extent this can be done properly portably. Utilities such as `tar` and `arc` provide a model for transporting hierarchies (directory trees), but some systems (e.g., MVS) have flat file stores that would prevent mapping the ER hierarchy conveniently onto the host file store on unpacking.

4.5.2 Portability considerations

The ER should contain only printable ASCII characters (apart from those delimiting record boundaries). It should contain *records* no more than 80 bytes long. The ER may contain either *fixed length records* or *variable length records* as appropriate for the target system. Where hierarchies are involved `tar` format should probably be used since a GNU implementation of the program is generally available. Where the ER is transported on magnetic or other storage media, the format of the media should conform to the ANSI standard for labelled tapes [9] or to the `tar` format [10]. Block lengths should be no more than 8 k bytes.

Requiring portability will obviously compromise what it is sensible to transport this way—it is not envisaged that a 4k×4k channel dataset would be transported in a text representation. The ER is aimed at applications like taking home saved copies of 'small' online histograms. The onus will be on the foreign site to do any necessary translation of the IR for very large datasets; the IR should be appropriately documented to allow this.

4.5.3 Ease of translation

The structure of the ER should be designed for easy interpretation and extraction of the data. An existence proof of this should be available in the form of standard Fortran77 routines for translating ER data types to Fortran arrays. The current scheme [8] makes life difficult by mixing a format largely compatible with Fortran 'free-format' with `CHANNEL` statements, which are not.

4.5.4 External representation operations

In addition to the operations on items detailed in §4.3, operations must obviously be provided for transforming data between the internal and external representations.

4.5.5 Window records

The ER for window records should be taken from the sort language definition [7].

4.6 Compatibility with present system

The new system will be required to run in parallel with the present one in some overlap period. This means that the new data base should be in a form that can be made to work with the old one, SDB [11], (if necessary by small modifications to SDB to avoid compromising the new design). This is another reason for requiring that the data base is implemented on top of the filing system.

Chapter 5

Displaying data

5.1 Data Boxes

Data are to be displayed in a number of *boxes*. There is a pre-defined limit on the number which may be displayed together, proposed as 12. *One dimensional* ((x, y)) or *two-dimensional* ((x, y, z)) data may be displayed in these. One-dimensional datasets (e.g., spectra, fits) may be overlapped in a box; two-dimensional!datatwo-dimensional data may not[1]. Other objects may be displayed in boxes, viz. *markers*, *windows* and *fits*. Boxes have attached *annotation* showing their limits (e.g., upper and lower channels) and the *name* of the *current dataset* or, if there is not a current one, the last selected. Overlapped datasets may be distinguished by colour or display style e.g., *histogram*, *dashed histogram*, *polymarker*. Boxes may optionally have *axes* (*annotated* in terms of channel number or an appropriate calibration). In the absence of axes, the minimum and maxmimum x and y values should be indicated. The name of the current dataset should be displayed in the box.

Various operations are required on the contents of the boxes. These fall into two categories: changing the display parameters (e.g., *panning* and *zooming*) and changing the data (e.g., overlapping or removing). When more than one box is displayed many of these operations are applicable either to a particular box or to all the boxes at once. Either type of operation should be easily selectable (preferably without having to change some 'option' in a special window, analogous to the present system). Some sort of *control panel* separate from the data boxes will be required for global operations and will have to fulfill some of the functions of a window manager.

5.2 Contents of data boxes

Boxes may contain the following objects. Those of different dimensions are mutually exclusive.

one-dimensional spectra usually displayed as histograms. These should be displayed so as to give an accurate impression of the local minimum and maxi-

---

[1] A decision is needed about whether spectra should be overlapped according to corresponding channels, as at present, or, more logically, according to corresponding calibration. In the latter case, it is either forbidden to overlap calibrated and uncalibrated spectra or one should assume that uncalibrated ones have the same calibration as, say, the last calibrated one to be added.

mum excursions in y—note that re-binning the data to one channel/pixel does not do so;

two-dimensional spectra usually displayed as grey-scale or false-colour images[2]. It may be necessary to re-bin them for display when more than one channel/ pixel would otherwise be displayed (c.f. one-dimensional histograms). The maxmimum size of a 2-d spectrum that can be displayed needs to be specified in terms of a sensible screen area and the screen resolution;

markers on channels of one- or two-dimensional spectra. These should indicate the channel, energy if appropriate, channel contents and marker type (current, peak marker etc.);

polylines (sets of arbitrary (x, y) points joined by straight line segments);

splines (as polylines, but joined with a spline instead of straight lines and possibly multi-valued);

polymarkers (points as for polylines but indicated with a *marker character* and not joined by lines; not to be confused with channel markers);

window regions as a set of markers in the one-dimensional case and as a closed polygon for two-dimensional spectra;

annotation in the form of axes, dataset names etc. to convey information about what is displayed;

fits Special types of one-dimensional spectra generated from the function fitted to a spectrum (§6.1.2.1) which are transient.

5.3 Box management

Data boxes are not independent (even if they are implemented as separate windows) but rather, subject to central control particularly regarding positioning when they are added or deleted. It should be possible to choose the *policy* for such positioning such that boxes either *tile* a fraction of the display or *overlap* (with the most recently selected ones nearest the top but with borders of all visible for selection). From the *box manager*'s point of view boxes differ only in their contents and position; they are initially displayed all the same size, for instance, but may be subsequently re-sized individually by the user.

The following operations are required on the box collection:

Add a box ;

Remove a selected box ;

Remove all boxes ;

Re-size all boxes ;

Move all boxes .

---

[2] Care is required regarding false colour. Medical physics wisdom seems to be that grey-scale is preferable to avoid a misleading picture. Certainly, arbitrary assignments of colours to contour levels should be treated circumspectly.

5.4 Box operations

The following operations are required to be available *local* to individual boxes and, where stated, also *global* on all the boxes displayed.

5.4.1 Tag channel

A mouse hit on a channel moves the *current marker* to that channel. The box and spectrum thus selected become *current* and should be indicated as such, for instance by a colour change. Criteria for *hitting* a particular channel with the mouse need to be experimented with: e.g., how close in pixels to the defined channel position is acceptable and how is a channel decided upon when more than one channel/pixel is displayed?

5.4.2 Region changing

Region changing should be available locally and globally, including:

Panning left/right and up/down;

Zooming in/out in x for one-dimensional boxes, in x and y for two-dimensional ones;

Expansion to a given x or y region;

Reset to the *natural expansion* for the box. This is defined to include the full range of all overlapped rather than just the first one added to the box as at present.

The amount by which panning and zooming changes the region (e.g., by a factor of two for zooming) should be resetable at the user's discretion. A *ring* of region specifications should be kept so that limited re-tracing is possible back through a set of changes to the displayed region. Separate rings should be maintained locally and globally. It should also be possible to store the current expansion specification separately from the ring and to retrieve it. Thus we also have these operations:

Revert to previous expansion on the ring;

Retain current expansion;

Restore retained expansion (not updating the ring).

5.4.3 Changing count range

Changing count range works differently for one- and two-dimensional data. In the former case, panning and zooming in y affects the range of counts displayed. Two other convenient functions are required on the y scale:

Times 2 *halves* the range displayed (so that the *data* appear to be magnified by 2);

Divide by 2 doubles the range.

The z display in the two-dimensional case is described in terms of *contour levels* (even if the data are not displayed in contour form). Appropriate operations on the levels are:

Specify contour levels Explicit values are given by the user for the number of contour ranges supported—eight is suggested as a sensible limit. The default for the contours is equally-spaced between the maximum and minimum z of the dataset displayed;

Contour up Only applicable if the contours are equally-spaced, in which case the spacing is subtracted from each level;

Contour down As for *contour up*, but adding the spacing to each level;

Times 2 Halve the value of each level;

Divide by 2 Double the value of each level;

Default contours Reset them to the default values.

5.4.4 Transformation

The x and y (and z where appropriate) scales of boxes may be chosen locally or globally to be either linear or logarithmic, subject to the logarithm of the data being well-defined.

5.4.5 Changing box contents

Overlap dataset/window Either the existing and overlapped items must be one-dimensional (spectra or graphs) or the overlapped item may be a two-dimensional window;

Un-overlap dataset/window The item to remove from the overlapped collection is chosen;

New data A single selected dataset replaces any displayed in the box.

5.4.6 Others

Markers should be able to be added to the display, removed from it and moved;

Information The parameters of the current dataset are displayed (e.g., its idxranges);

Remove fits All fits and markers associated with fitting are cleared.

5.5 Display dependencies and refresh

It should be possible to define a *dependency* for each data box such that when a component of the dependency changes the box display is updated accordingly. Typical dependencies might be of a slice upon the position of a window or of a *linear combination* of spectra upon the coefficients.

A special case is of a live (online) spectrum upon a *timer* to implement an automatic *refresh mechanism*. This mechanism should be available globally on the data boxes, overridden locally with a specified period (which may be infinite). It should be possible for data with another dependency to be auto-refreshed. The user should be able to trigger a refresh independently of the timer.

Chapter 6

Data analysis and manipulation functions

The following operations on data should be provided in addition to those listed in §4.3, 4.5.4.

6.1 One-dimensional spectra

6.1.1 Creating new spectra

These operations (which create new spectra rather than over-writing a spectrum that is altered) must propagate the error spectra appropriately and write 16- or 32-bit results as appropriate. Any precision or mixture thereof is acceptable on input. New spectra have a name generated according to the currently-selected *policy* and a title which indicates the input spectra/spectrum and the operation performed.

Linear combination of an arbitrary number of spectra as provided currently by `ADDSEQUENCE`;

Re-scaling A linear transformation of either the x or y scale. The former means *re-binning* the channel contents smoothly, the latter merely changing them. Currently provided by `XSCALE`, `YSCALE`;

Editing Either:

> *Content changing* A range of channels may be set to have all the same value (occasionally useful e.g., where overflows have occurred). Currently provided by `CHANGECHANNELS`;
>
> *Range changing* The channel range may be truncated or expanded. This is not the same as x-scaling since channels out-of-range are ignored on truncation and extra channels filled with zeroes on expansion.

Smoothing using binomial weighting as currently provided by `SMOOTH`;

Other operations … like unfolding, non-linear re-scaling (e.g., square root)?;

6.1.2 Analysis

The functions below should optionally write results to the *journal file* (§9.2) as well as displaying them on the screen.

6.1.2.1 Background and peak fitting

The *background* in a selected set of regions may be fitted by various sorts of functions which are supposedly physically meaningful in different situations. Parameters of one or more *peak*(s) may then be determined in another region relative to this background, either by fitting a function or by a simple calculation like that used in *integration*. Peak and background regions must all be disjoint. The possible types of background fit function are:

None A degenerate case for use when there really isn't any background or to allow the peak fit to do all the work;

Flat ;

Sloping ;

Polynomial of order 0, 1, 2 or 3;

Exponential .

Polynomial and *exponential* fit the same function in all the regions. *Flat* and *sloping* fit an appropriate linear function independently on each side of the peak region and extrapolate it to the edges of the peak region. They have two varieties:

Line where the background function used in the peak region is obtained by linear interpolation of the extrapolated values at the peak region boundaries;

Smooth where the interpolation is done smoothly in this region, the value being determined by a running integral over the peak area above the background starting from the lower limit of the peak region. This is appropriate in cases where the background below a peak is heavily influenced by the peak's intensity.

Peak fitting functions may be either those supplied by the system or defined by a user-supplied routine that must be linked with the fitting program. User-supplied functions may be non-linear functions of their parameters. The system-supplied ones are of two types:

- a summation of Gaußians (one for each peak supposed to be present) with optional *skew* to account for tails on peaks due to detector imperfections. The parameters (centroid, width and skew) for each peak may fixed or fitted and the peak area calculated. The fit may be constrained to have the same parameters or the parameters varied for each independently. If the centroid values are to be fitted initial guesses are either supplied by the user or chosen rather arbitrarily by the program. Width guesses are supplied automatically;

- an exponential folded with a Gaußian[1].

Other commonly-used functions may be supplied according to demand (e.g., optical lineshapes?).

Fitting produces temporary *fit objects* corresponding to the fitted functions for the peak and background in the regions over which they are defined. These will be overlapped on the display to indicate the quality of fit achieved. They may be saved as permanent spectra and subsequently used, for instance, to generate background-subtracted spectra.

Fitting is only possible if the spectrum concerned has associated *errors*, which determine the χ^2 and error estimateserror estimates on the fitted parameters. The latter should not include contributions from the background fit since they are independent. Area estimates extracted from the fits are supplied asis and (for spectra with an efficiency calibration) also efficiency corrected using the efficiency appropriate at the centroid.

Features described here are partly as provided by the current `BACKPEAK` program, but this needs re-writing. The functionality of `DECAYS` is incorporated into the system-supplied functions.

6.1.2.2 Other

Integration A simple calculation of peak and background areas, peak centroid and width between two channel limits as currently provided by `INTEGRATION`;

Automatic peak estimation The current `PEAKFIND` program, or an equivalent, should be available to provide estimates of peak centroids and areas, particularly for use with labelled plots;

Peak identification Facilities for identifying peaks in reaction spectra with particular γ-ray or charged-particle decays should be provided. These will rely on appropriate nuclear data bases like the current `GAMMADB`;

Peak-to-total calculation for easy determination of the quality of Compton suppression in γ-ray detectors as per `PEAKTOTOTAL`;

Escape peaks identification (peaks 511 or 1022 keV below another) as per the present `ESCAPES`;

6.2 Two-dimensional spectra

Slicing should be possible by *projecting* the data onto the x and/or y axes in the region specified by a window. A special case where the window is the whole displayed region should be allowed;

Integration of data in a region specified by a window. Details need to be worked out (e.g., how to define a background, if at all);

Re-scaling ;

---

[1] Fitting sums of exponentials is a poorly-defined problem [6], so fitting an exponential with exponential background should not be allowed.

Linear combination .

6.3 Graphs

Fitting with user-defined functions as for spectra (without consideration of a background);

Interpolation It should be possible to interpolate a graph's points, for instance to form a histogram from it;

Combination It should probably be possible to combine graphs in linear combinations, for instance. This will require definition of what happens when the x values don't coincide.

6.4 Maps

As for graphs.

6.5 Windows

Windows should be editable by moving the component markers in the one-dimensional case and by moving the vertices with *rubber-banding* in the two-dimensional case. Creating a window from scratch is a special case.

Chapter 7

User interface principles

This chapter discusses some principles which the implementation of the user interface should follow. These are based on general interface wisdom, experience with the current system and some attempted insight. It is important to get the 'feel' of the system right, since this is what makes it usable. Unfortunately, this is just the area that is hardest to get right! More principles than thought of here should emerge on detailed specification or implementation. It is debatable whether these issues are requirements or belong in a design document, but they are included as they (should!) have a big influence on how users perceive the system. A useful user interface meta reference is [12].

7.1 Meta-principles

The aim is that the system will be able do what a sophisticated user would expect and in a way that is comfortable for the naïve. As previously stated, we have to aim at a wide range of users with different degrees of sophistication and experience. However, we should regard them as being technically competent and not feel it necessary to nurse-maid them to the extent that might be necessary with commercial applications aimed at general users. (That would be too time-consuming anyway.) The primary aim is *not* 'user-friendliness' but rather an interface that is powerful, flexible, consistent, maintainable and, perhaps, hard to go wrong with. Subject to that it is, of course, intended that it should be as easy as possible to for the intended users to learn and use.

Principles that are decided on (e.g., the ability always to be able to do with a command language what you can do with direct manipulation) should be made explicit in the documentation and adhered to consistently. Any exceptions would need a good excuse and would need to be pointed out carefully where they occur. I suggest a touchstone for whether things are right is the complete operation of doing and displaying the results of a peak/background fit with several fitting regions and a few mistakes.

7.2 Use of display facilities

7.2.1 Colour/shading

The requirement to support monochrome displays restricts an implementation in its use of colour and shading. Choices should be made only from the set known to be sensibly (distinguishably) rendered (by dithering) on the hardware targets. Such rendering, however, is often going to fail (notably with one-pixel-thick lines), so that distinctions by colour on the display should not be relied upon. Attention should be paid to common colour blindness—we have a test subject! Since dithering slows the display, it may be useful to allow the user to turn off attempts to render it on monochrome displays.

7.2.2 Sound

Platforms such as the SparcStation would allow use to be made of *sound*—even in stereo—for instance to alert the user to exceptions. It is possible to envisage applications like 'recorded messages' giving warnings or explanation, rather than disrupting the screen interaction with visual messages. This is, however, regarded as bad design in systems designed for 'public' use and should be avoided unless users are willing to experiment with using headphones. Sound should be restricted to beeps in the case of error conditions (e.g., to warn that the operation attempted has been ignored) and perhaps sounds of greater insistence and persistence for serious exceptions reported at a display running an experiment (§9.3).

7.3 Commands/Icons

Commands and icons should be chosen carefully to avoid confusion—in particular, confusion with a destructive operation. Thus *delete-* or *clear-*type commands or buttons should be well isolated syntactically or spatially from others. (There are counter-examples in the current system.) The sensitivity to such confusion has to be determined partly in the light of user feedback. Icons should have an obvious significance (by common consent of users—which probably rules out most pictures!), otherwise a textual label should be used.

7.4 Interaction in general

The principles discussed in some sections below are clearly interrelated.

7.4.1 Pre-emption

Pre-emption of the user by the interface involves the user being forced into a certain course of action at some time e.g., having to answer a question in a dialogue box but being unable to access the tool to provide the information. This tends to happen in the present system due to its single-tasking nature. It should obviously be avoided. Sensible use of multi-tasking in the windowing system will help to avoid it. Avoiding pre-emption is a reason for having a multi-tasking windowing system.

7.4.2 Modes

Modes should be avoided as far as possible. They tend to lead to confusion, especially where there are subtle bugs in the interface logic, as has sometimes become apparent in the present system. In some situations they will be helpful, however (see §3.4). Modes can lead to pre-emption.

7.4.3 Hidden information (WYSIWYG)

For naïve users, at least, it is helpful for the interface to adopt *WYSIWYG* ('what-you-see-is-what-you-get') tactics as far as practicable. Thus, for instance, appropriate operations should be visible as menus, buttons, etc. and pop-up menus are discouraged, definitely if they aren't indicated by a change in the cursor appearance, and likewise with modes (§7.4.2). This principle may be relaxed substantially for expert users since WYSIWYG also equates to 'what-you-see-is-all-you-get' and may hinder effective use. (This is a somewhat unconventional interpretation of WYSIWYG.)

7.4.4 Proper determination

It is necessary to strike a balance between over-determination, where the user is trapped into rigid dialogue (probably connected with pre-emption) and under-determination, where the user doesn't know what to do next. Having modes for different levels of experience helps here (§3.4).

7.4.5 Direct manipulation and scripting

To be more specific about the requirement (§3.1) for direct manipulation and scripting, we may adopt the principle that any DM operation should have a direct equivalent in the scripting language.

7.4.6 Least astonishment

Design decisions should be guided by the principle of *least astonishment*, such that the interface's behaviour is intuitive (§2). Where in doubt the behaviour should be 'canonical', which probably means following the old system or a commonly-used tool.

7.5 Misleading information

The interface should not display misleading information, encouraging the user to invoke an operation or select data that would be inappropriate. Thus, for instance, if a 2-dimensional spectrum is displayed, the menu item for the *overlap* operation should not be selectable; if a 1-dimensional spectrum is displayed, *overlap* should present the user with a selection of items to overlap, all of which are appropriate (not 2-dimensional, for instance). The idea is to avoid as much as possible the situation in the present system of selecting something and being told it was wrong subsequently.

7.6 Speed

An estimate is needed of speeds acceptable to users for representative operations to guide the implementation (if the requirements can actually be satisfied!).

Chapter 8

Presentation graphics and hardcopy

8.1 Graphics editor

An *object-oriented graphical editor* is required for composition of *presentation graphics*. This should provide the canonical operations of such tools (modelled on MacWrite) and others as necessary to tailor it to the types of graphical object used in NIGS (e.g., spectra, axes). It should be possible to *cut and paste* objects from the normal visualisation display into the editor and manipulate them directly to compose the required picture. This might typically be a collection of labelled spectra, perhaps with fits and a graph derived from them. Modes should be available for specialised functions like drawing *decay schemes* along the lines of the `LEVELS` program in the current system. As well as composing pictures interactively, it should be possible to write commands in a language designed to achieve the same result so that pictures may be generated automatically and subsequently edited interactively.

Such a facility does not currently exist. Note that a command-driven system like that provided by PAW [3] is not proposed, but rather one like `fig` [13] with a direct manipulation interface and a high level intermediate graphics representation. Some examples of the sort of operations required are provided by the facilities of the current `PSSKETCH` and `gnuplot` program [14]. Pictures should be composable by direct manipulation and by `PSSKETCH`-type commands. This functionality and the requirement to cope with objects like spectra precludes using a commonly-available tool (like `fig`) to do the job, but it may be sensible to extend existing facilities rather than producing another from scratch. One possibility would be to merge something `fig`-like with something `gnuplot`-like and cope with arbitrary data objects by providing a mechanism to pipe any data file into the program through an arbitrary filter.

8.2 Hardcopy

Hardcopy is required both of the presentation graphics and plots of data in a different form for visualisation purposes. The latter will typically need to be high resolution plots of spectra with axes labelled by channel and energy (at a size and frequency that would not be appropriate for presentation graphics) and with peaks automatically labelled (all as currently provided by the `SPECTRA` program). Complete dumps of the display are required and dumps of part of it (e.g., only that relevant to data-taking and analysis) should be available. As stated in §A.1.1 all such graphical output should ultimately be available in PostScript form.

Chapter 9

Miscellaneous functions

9.1 Dataset selection

Some care needs to be devoted to the part of the interface that selects datasets from the data base for display or other operations. Subject to the uniform structure based on the filing system which has been advocated earlier, this is essentially a file-selection facility. The files available for selection for use with an operation should only be those appropriate to the operation.

A mode of the file selector might, however, be provided to give information (like their titles and sizes) on all files in a directory as currently available. Manipulation functions (copying, re-naming etc.) as described in §4.3 should be available from the file selector. These information and manipulation functions should be available at any time, particularly between the file selector being displayed and a dataset being selected. It should be possible to move quickly between a number of previously-visited points in the directory hierarchy.

9.2 Journalling/logging

The facility should be provided to record two different types of transcript information from the session in the form of a

Journal intended to record the results of analysis operations (§6.1.2);

Logbook intended to record information during an experiment, including:

- error messages;
- notes (general explanations/excuses etc. that you might write in a run book);
- saves and deletions of online spectra;
- tape mounts and file opens;
- stops and starts.

Logbook entries should be time-stamped.

Separate streams should be available to record both types of information and any application should be able to copy its output to either. (It is expected that all such output will be a *copy* of information displayed on the screen, or at least a paraphrase of it.)

9.3 Exception/warning reporting

Exceptions (errors) and warnings may arise either within NIGS itself or be reported to it by another system to which it provides the interface (typically to do with experiments).

9.3.1 Internal

Exceptions arising from operations within NIGS should not be logged unless they indicate system malfunction (see §9.4). If modes are available for different levels of expertise they should be able to report exceptions differently. An expert mode will give a brief indication of the error—perhaps just a beep—without holding up further work, whereas a novice mode is likely to put up a dialogue box giving an explanation of the exception and requiring action to proceed. Although the way reporting is done may be different depending on whether it arose from direct manipulation or a script, it should be done consistently in each case.

9.3.2 External

External exception events arise asynchronously. NIGS must 'listen' for them and take appropriate action when one occurs. They are expected to be generated by an external *exception server* (it's not supposed actually to generate exceptions itself!). The user must be alerted to them, depending on their degree of urgency (which information the server should provide). Various schemes for doing this are possible and probably different ones are appropriate for different severities. For severe errors probably a window should pop up (away from the area the user was just about to hit with the mouse!) and there should also be a suitably insistent audible indication (more than just a beep) if the workstation can generate it (Sparcs can). The user does not then actually have to be looking at the display or awake to be alerted. Error reports should be entered in the logbook journal file. Where there are multiple outstanding messages, these should be scrolled so that the user can view them all, up to an appropriate buffer limit. Having looked at them, the user should be able to clear this scroll area/buffer for when more arrive. Some mechanism has to be provided to avoid being swamped by huge numbers of messages if something is badly broken, but probably this is the server's job.

It will probably be helpful to have some semi-expert system to help with diagnosis of fault conditions from the exception report(s). Again this is probably the server's ultimate responsibility, but NIGS will need to provide a suitable interface to it.

External exception reports should be made clearly distinguishable from internal ones.

9.4 Bug reporting

A built-in facility to report *bugs* should be provided as done currently with the `HARDWARE` and `SOFTWARE` report facilities. This should be done by automatic e-mail to the maintainers. In some cases it may be possible for the system to detect fault conditions internally and attempt to dump the relevant context and forward it to the maintainers. In any case, the bug-reporting facility should be integrated where possible with error reporting (§9.3) at least to the extent that the user making the report doesn't have to copy down details of the last error message but, rather, can retrieve it from some buffer. Version numbers of programs should be readily available (preferably generated automatically from the last error report).

Note that 'bug' ought to include commonly perceived mis-features which—like the placement of objects on the screen, for instance, slow down users or make them error-prone.

9.5 Instrumentation

It will be helpful if *instrumentation* can be included in the system so that information on its use can be extracted automatically to provide objective guidance in making improvements. Such information might include the frequency of use of commands, most common settings of options and logs of particular error messages.

Chapter 10

Documentation and help facilities

10.1 Overview

The information about and inside NIGS must cater for the range of users it is expected to have (§7.1), in other words it should be usable at a variety of levels with differing degrees of difficulty as far as possible. The complete tyro needs to be able to get going quickly and find the way around as necessary; experts need to be able to remind themselves about more esoteric aspects equally quickly and easily. Obviously catering satisfactorily for the complete range of users is a difficult problem, but one which needs to be attempted.

As far as possible, the system is expected to be *self-documenting* so that either the way it works is obvious or it is simple to get it to tell you the way it works. The present system attempts this with certain elegant features, but these have not been well-maintained latterly. It is necessary also to have offline documentation so that one can learn about the system without sitting at it if necessary.

A good example of the general facilities one should try to provide is given by EMACS 'the advanced, self-documenting ... editor' [15]. Another example is provided by The Publisher [16].

It is important to maintain *verity* in the information facilities—they should be updated and easily updatable whenever changes are made. As far as possible this should be automated and facilities to allow it built into the system.

10.2 Online documentation

The online documentation facilities needed are of two types, although it is preferable to derive them from the same source: an *online reference manual* and *context-sensitive help*. Online documentation should be such that there is no need to consult the printed manual when at the terminal.

10.2.1 Manual

The manual gives a general view of facilities and should allow one to find the appropriate strategy for doing something without having to work one's way down a tree of menus etc. to find something that looks as though it might do the job. The manual is expected to make use of *hypertext* facilities (to make following cross-references easy) and appropriate pictorial material, display of which the hardware will be able to support. This can probably be done using TEX as the formatter with suitable mark-up and a hacked previewer.

10.2.2 Context-sensitive help

It should always be possible to get help on the various ways of doing things in a given context even if menus, icons etc. are supposedly intuitive[1]. Thus explanation of selectable items and an indication and explanation of allowable alternatives where typed commands are required should be available. This might consist of a link into the hyper manual. Graphics should be available in this situation to allow the best explanation of the interface. Any exceptions to established principles (§7.1) must be obvious in context.

10.3 Offline documentation

The offline documentation is expected to be derived (printed) from the same source as the online one in the same way as the GNU *texinfo* system so that the only significant difference between them is the hypertext facilities and possibility of colour, motion etc. in the online version. The offline version is required so that people may learn about the system without sitting in front of it.

10.4 Tutorial system

A tutorial system is helpful to new users and should be provided, although at lower priority. It should guide new users through the facilities they are most likely to use (sufficient to be able to cope with a night shift!). The tutorial might usefully be extended to cover things other than the graphics system itself, principally the mechanisms for controlling experiments.

---

[1] It should be remembered that supposedly 'friendly' systems like the Macintosh can be quite baffling even to sophisticates when they encounter them for the first time and aren't tuned in to the twisted mind of the icon designer!

Appendix A

Constraints

A.1 Hardware and software platform

This section covers some of the *non-functional requirements*[1] of the new system dictated by the current state of technology and finance and enshrined in recent policy statements [4, 17].

A.1.1 Hardware

The only realistic cost-effective hardware platform at present is in the form of *graphics workstations* as provided, for instance, by Sun Microsystems. (There are competing systems from DEC, Hewlett Packard etc.) These provide both the cpu power and the graphical display capabilities needed. They are expected to operate in a distributed environment, sharing resources such as mass storage over a *local area network*. Alternative approaches based on central systems driving X/NeWS terminals, for instance, do not seem realistic or cost-effective at present.

We take the Sun SLC1 as defining the minimum sort of hardware facilities available for systems on which NIGS should run. This means:

- ~10 MIPS, ~1 MFLOPS cpu power;
- 'sufficient' real memory (≥12Mb) to support a windowing system, application software and significant amounts of data related to the display!characteristics;
- a monochrome screen of resolution ~1000×1000 pixels (although colour displays are expected to be the norm);

A graphics accelerator is not assumed. The performance of the LAN and file server is assumed sufficiently good that swift file access is not a problem.

Hardcopy devices for use with the system are expected to support POSTSCRIPT primarily. There should be a means of translating any output meant for hardcopy to POSTSCRIPT if it is generated in another form. The next release of GNU `ghostscript` is expected to be good enough to drive devices that don't support POSTSCRIPT directly (e.g., the HP PaintJet colour plotter) should the need arise.

---

[1] 'Non-functional' requirements define constraints under which the system should operate [1].

A.1.2 Software

A.1.2.1 Operating system

The only reasonable type of operating system to consider is *Unix* (or 'Unix-like' e.g., Mach) even if one could run something else on the hardware provided. There are many varieties of Unixes and where it is important to distinguish between them one should insist on Posix-compatibility[2] [18]. This will be capable of distributed operation, at least to the extent of a distributed filestore based on NFS.

A.1.2.2 Languages

Application software should be written in standard C and/or Fortran or some other language for which freely-available, portable implementations are available, like `awk`, Scheme or shell scripts, for instance. The standard for C should be the ANSI one [19] where this conflicts with the de facto 'Kernighan and Ritchie standard'. (The ANSI standard is supported by the freely-available GNU `gcc`.) The standard for Fortran should be Fortran77 [20] since solid Fortran90 compilers are unlikely to be quickly available. We assume that appropriate libraries can be licensed where appropriate (for instance NAG for numerical analysis [6]).

A.1.2.3 Windowing system

The graphics system is expected to be built on top of a multi-tasking, *WIMP*-orientated windowing system. The choice is probably between X windows and NeWS [21] but the choice between them is an implementation decision. (A requirement like the ability to cut objects and paste them to the presentation graphics editor (§8.1) would, however, appear easier to implement in NeWS, as would general communication between independent applications running in their own windows.)

A.2 Distribution

The system will operate in a *distributed environment*. For analysis/visualisation purposes it is expected that usually only the data will be non-local, not application programs—the workstations upon which the system runs are expected to provide the necessary computational ability (§A.1.1)—but the ability to run application programs on a system remote to the graphical interface should be provided. For apparatus control etc. it will be necessary to communicate with remote processes. Ideally it should be possible to distribute components across the *wide area network*. This is expected to be particularly useful for remote experts to diagnose and fix problems with the local system (especially apparatus).

---

[2] Posix is a standards effort in this area. [Is SunOs Posix-compliant?]

Bibliography

[1] Ian Sommerville. *Software Engineering.* International Computer Science Series. Addison Wesley, Wokingham, England, third edition, 1989.

[2] Darrell C. Ince. *An Introduction to Discrete Mathematics and Formal System Specification.* Oxford University Press, Oxford, UK, 1988.

[3] R. Brun et al. *PAW: Physics Analysis Workstation*, 1989. CERN program library entry Q121.

[4] E. Owen. Data Acquisition and Electronics Estimates and Forward Look (91–96). Data acquisition working party document, 1990. ECGO/8–25.

[5] C. D. Wooff. *Nucl. Instrum. Meth.*, 214:419, 1983.

[6] NAG Ltd. Nag numerical analysis library.

[7] Liverpool software systems group. EUROGAM *Project: Sorting Language.*

[8] NSF Software Systems Group. *NSF Data Handling System: Spectrum Database.* Edition 2.3.

[9] Ansi tape labelling.

[10] Jay Fenlason. `tar`*: The GNU tape archive.* Free Software Foundation, Inc., 1990.

[11] Introducing the SDB spectrum access system. NSF Software Systems Group. Edition 1.2.

[12] Harold Thimbleby. *User Interface Design.* ACM Press Frontier Series. Addison Wesley, Wokingham, England, 1990. ISBN 0–201–41618–2.

[13] Micah Beck and A. Siegel. Transfig: Portable graphics for TEX. TUGBOAT, 11(3):373, 1990.

[14] Thomas Williams et al. *GNUPLOT: An Interactive Plotting Program.* The Free Software Foundation, Cambridge, Mass., USA. Version 2.0. Available from GNU depositories (see [15]).

[15] Richard Stallman. *GNU Emacs Manual.* Cambridge, Mass., USA, fifth (Emacs version 18) edition, 1986. User guide to the Emacs editor, an example of a large, customisable system worked on by people all over the world using various good interface techniques. See also the manual for its programming language. Part of the Emacs distribution from GNU depositories like `prep.ai.mit.edu`.

[16] ArborText inc. The Publisher, version 3.0.3. Available on Daresbury Suns.

[17] V. Pucknell. University grants 1991/1993. Data acquisition working party document, September 1990.

[18] Posix reference.

[19] Brian W. Kernighan and Denis M. Ritchie. *The C Programming Language.* Prentice Hall, second edition, 1988. ISBN 0–13–110362–8.

[20] American National Standards Institute. *American National Standard: programming language FORTRAN*, 1978. ANSI X3.9–1978 ISO 1539–1980(E).

[21] *X11/NeWS Version 2 Server Guide.* Part Number: 800–4898–10. See also references therein.

Index