University of Glasgow
Department of Computing Science
Lilybank Gardens
Glasgow G12 8QQ

University of St. Andrews
Department of Computational Science
North Haugh
St Andrews KY16 8SX

Using a Persistent Environment to Maintain a Bibliographic Reference Database

Persistent Programming
Research Report 24

A2.

Using A Persistent Environment To Maintain A Bibliographic Reference Database

Cooper, R.L., Atkinson, M.P. and Blott, S.M.

Abstract

We claim that a persistent store is ideally suited to the development of software which supports the production of all kinds and sizes of document. The storage within the same space of the text and diagrams of papers, a body of references and all the software to maintain them provides an extremely powerful environment for developing both the system and the documents. We envisage such a system could encompass, among other functions, word-processing, diagram manipulation, automatic bibliography construction, the production of indexes, page make-up and the maintenance of mailing lists. A persistent environment has the particular merit of facilitating the replacement of code, so that improved versions of tools can be easily inserted and more than one tool could be provided for any function. The user can pick a favourite word processor or choose a simple one for a simple job, turning to a more powerful one where necessary.

As a start to developing such a system, we have implemented a set of programs which automatically build the bibliography at the end of a paper in a manner similar to the Scribe system. A variety of procedures are available to produce the bibliography in a number of different text processing languages. The database also holds the formats required by a number of journals and there are facilities for maintaining this set of formats. The method for incrementally adding compiled code, for generating different formats and supporting a variety of hosted text processors, illustrates a significant ability obtained by using first class procedures in a persistent store.

Introduction.

When producing software for document production, a number of problems arise:

the conflict between ease of use and fine control over the structure of the document;

the past history of users of other products who want to be able to use all the features to which they are accustomed;

the fact that the document's layout may need to be completely re-shaped, while its contents remain the same;

the desirability of linking together the various components of a document, systematically and flexibly.

A persistent environment assists in the solution of these problems. Versions of software may be provided with equal availability so that the user can choose the most appropriate or the most familiar. Software can be provided which presents a different "view" of the same document to vary its presentation. It also becomes easier to provide facilities for mixing classes of data in a system which allows any data structure to be stored. For instance, it is simple to describe the insertion of a diagram into a piece of text in a more meaningful way than the MacIntosh Clipboard [APPL84], for instance, which merely copies the component and keeps no dependency record.

The development system we have chosen for our experiments in document production is the PS-algol system developed at the Universities of Edinburgh, Glasgow and St. Andrews [PPRR12, ATKI86a and ATKI86b]. PS-algol is a strongly-typed block structured language, which supports persistence, has excellent graphics facilities [MORR86] and allows procedures to be first-class objects [ATKI85].

Data and programs are made to persist by putting them into structures called "databases". These are tables of key/result pairs, where the key can be any string and the result can be a structure of any complexity, referenced through a universal pointer. The result pointer may, for instance, point to another table or to all the information about a reference or to a packaged procedure. Note that every object in the database is fully typed so that data cannot be misused. Accesses to a pointer value constitutes an implicit, statically determined projection from the universal union to a specific type, akin to Cardelli's type **dynamic** [CARD84]. The remainder of the program can be entirely statically type checked. It is strongly typed throughout. Discussion of this form of delayed checking and a comparison of alternatives may be found in [ATKI87 and COOP87b].

The power of the mechanism lies in the fact that if an object is explicitly placed in a database, then, at committal time, every other object that it refers to is dragged into the database with it. For instance, a reference which is made up of an author field, a date field, etc., is stored by entering a pointer to the reference into the database. and this will automatically bring in all the fields of the reference. Thus the structure imposed on the data to make it meaningful and manipulable by the program is also used as the structure in which it is stored. Figure 1 illustrates the ability of the

Persistent Store to hold a variety of different objects. It shows a map of the objects stored in the bibliographic database including: some utility procedures; help information; procedures for producing output in various formats; structural information about the references; and the references themselves.

The graphics facilities are also valuable in building a usable system. A good human-computer interface is an essential component in any piece of modern software and often this can only be achieved by glueing low-level routines into the program. This leads to unnecessary complexity and to machine and device dependence. PS-algol has two types specified for graphical work: the **picture** which is built out of points in a real Cartesian space and lines joined recursively; and the **image**, which is a rectangular array of pixels. Operations are available which scale, translate and rotate pictures and do raster operations on images as well as operations to produce images from pictures and strings. There is also a mechanism for creating pop-up menus and an interface provided for a mouse.

In PS-algol, procedures are first-class objects. This means that they can be passed around as variables or parameters of procedures and they may be stored in the persistent store in just the same way as any other object. Therefore it is a simple matter to store a procedure to perform a given function and then later replace it by an improved or different version. It is possible to store a parallel set of procedures and then to provide a menu for selecting which is to be used at a given time. There is also a version of the compiler which may be called during the execution of the program and so it is possible to create appropriate procedures as they are needed and compile them on the fly before storing them for later use.

This paper will describe how these facilities are used to build a system which stores bibliographic references and uses them to build the bibliography at the end of a paper, automatically. We will describe how all this information is stored in the database, will then describe the software provided and finally give a description of how the system is used.

System Overview.

The Bibliographic Reference Database Program (BRDP) manipulates **references**. A reference may be regarding as consisting of the following information:

- the **type** of reference it is - whether it is a book, a paper, etc.;
- a **citation key** for insertion into the paper to be processed;
- a set of **fields** and their values, such as "author", "title", etc.;
- a set of **key-words**;
- an **abstract** of the paper;

and - ultimately the **text of the paper** itself.

Our present system only makes use of the first three of these. The set of available Reference Types is derived from the Scribe Manual [SCRB84]. Associated with each Reference Type is a set of fields which are essential to specify that kind of publication, as well as a set of fields which may optionally be present. For instance a reference of the "Article" Type must have a pages field, but one of the "Book" Type need not. The way in which a reference will be layed out in the bibliography produced by the program will vary from Type to Type and so each Type has two associated layout specifications. One of these describes the way the citation keys will appear in the final text and the other describes the way each reference in the bibliography will be layed out.

Like Scribe, we support a number of "Reference Formats", which determine the preferred layout styles for various publishing organisations, such as "IEEE", "CACM" and "SIAM". For each of these Reference Formats, the Reference Types available and the required fields and layout specifications for those Types may vary. Each Reference Format also contains a specification for the order in which the references will appear in a bibliography - for instance alphabetically on author name or in the order in which they are cited in the paper.

To support this taxonomy, the BRDP maintains the following structural data:

- a set of all the field names known to the system;
- a set of all the Reference Types known to the system, with default values for the fields required for this type and a layout specification.
- a set of all the Reference Formats known to the system, each containing a set of Reference Types and a sort order specification.

The references are divided into Topic areas for storage. The BRDP maintains a set of such Topics, each of which contains a set of abbreviations, a set of references and a set of all the authors in the set of references. The abbreviations consist of pairs of strings containing the short and long forms of the abbreviated string. They are used to shorten the amount of data which needs to be entered and stored in the table of references. Within this table, the long form of any abbreviated string may be replaced by "@value" followed by the short form. This data, together with the structural data and the program modules are stored in a database in the manner shown in Figure 1. The organisation of this database will be described in more detail below.

The BRDP supports the following functions:

- a facility to set up a fresh database;
- editors for each of the sets of structural data;
- an editor for the set of Topics, enabling Topics to be added and deleted;
- an editor for the abbreviations in a Topic;

- bulk load and bulk dump facilities for a Topic;
- a facility to browse the references by author name;
- an editor for the set of references in a Topic;

and - a facility for creating the bibliography for a paper.

The latter reads through the text of the paper replacing the following:

- **@cite** followed by a citation key in brackets is replaced the citation key in the layout required by the chosen Reference Format;
- **@partbibliography** is replaced by the list of references since the last @partbibliography or the start of the text - the layout of the references and the order in which they appear also depend on the Reference Format;
- **@bibliography** is replaced by the list of all the references since the start of the text.

When creating a bibliography, the user specifies the following:

- the **Reference Format** to be used;

and - the **Output Medium** to be used.

The former determines the layout of the final document, by referring to three strings:

- the **sort order** asssociated with the Reference Format. This consists of a series of letters each specifying which field next to sort the references on. For instance "AY" means sort first on author, then on year.
- the **key layout** associated with the Reference Type within this Format. This is a set of strings concatenated with the following structure:
  - "@" followed by a field name means print the value of the field;
  - "#" followed by a string means print that string;
  - "'n" and "'t" mean newline and tab, respectively.
- the **reference layout** associated with the Reference Type within this Format is a string structured in the same way as the key layout.

The Output Medium determines the way in which the output will be produced., whether to the screen or to a text file or to a file suitable for input in a text processor, like TEX [KNUT84], for instance.

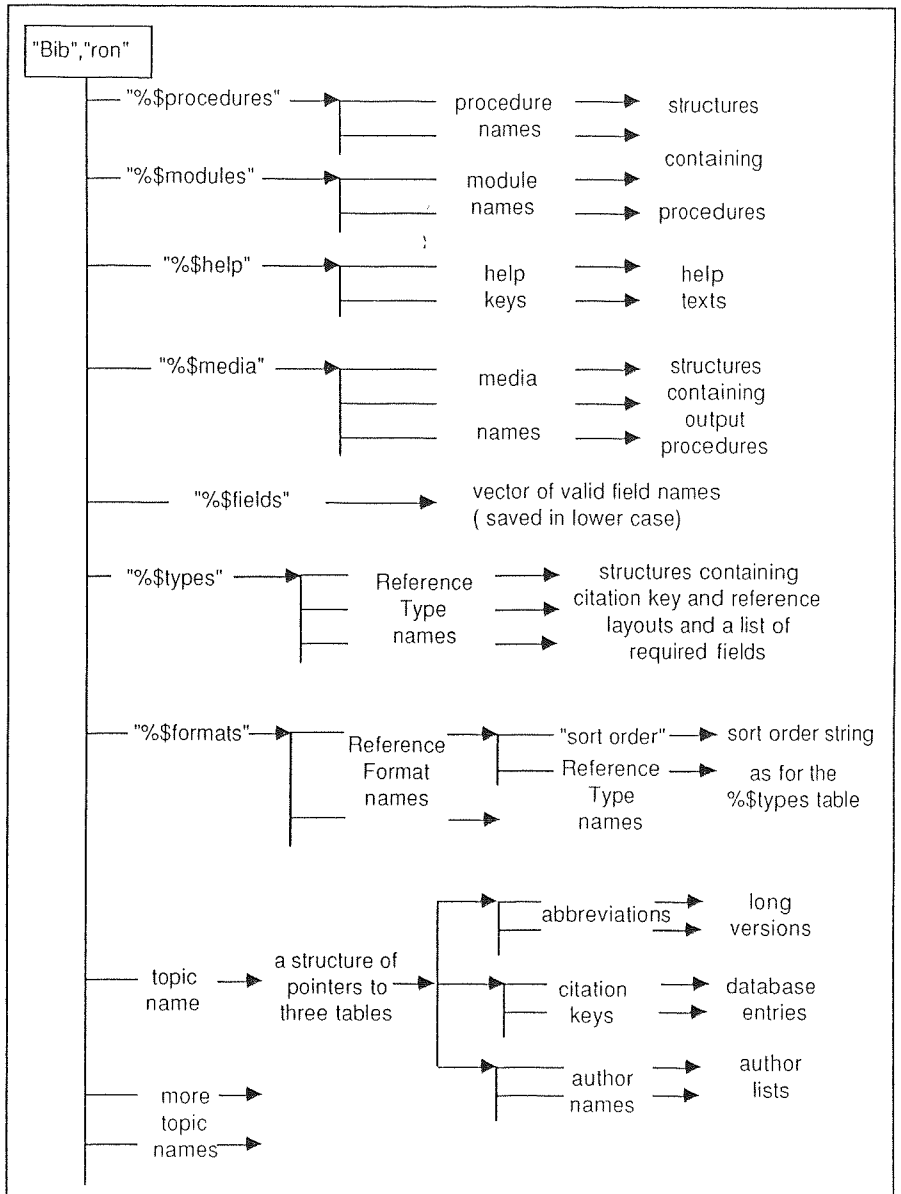

Figure 1. The Bibliographic Database Layout.

A Bibliographic Database.

The layout of the database used by the system is shown as Figure 1. The top level table of the database contains the following:

- A table of utility procedures, keyed by "%$**procedures**". Each utility is placed into the table, keyed by its name. The table is organised in the systematic way described in @cite[COOP87a].

- A table of the larger modules that are called from the initial menu, keyed by "%$**modules**". Each has structure, **proc( )**, and is placed into the table, keyed by its name.

- The help information, keyed by "%$**help**". The text for each help screen is stored with a string key which the program uses to find it.

- A table of post-processors, keyed by "%$**media**". Each one is keyed by a recognisable name, like "TEX" say, and consists of a package of five procedures which:
  - initialise the scan of the paper;
  - scan a single line of text;
  - initialise the bibliography output;
  - finish the bibliography output;

  and • finish the whole process.

- A vector of all the valid field names known to the system, keyed by "%$**fields**".

- A table of all the Reference Types known to the system, keyed by "%$**types**". Each entry in this table is keyed by a Type name, e.g. "book", and contains default values for this Type of reference. The information stored is:
  - a string which defines the citation key layout;
  - another defining the reference layout;

  and • a vector of the names of the required fields.

- A table of all the Reference Formats known to the system, keyed by "%$**formats**". Each entry is keyed by a format name, e.g. "IEEE", and points to a table, which contains the bibliography sort order, keyed by "**sort order**", and an entry for each Reference Type known to this format. These entries have the same structure as those in the "%$types" table. When producing the citation keys and references in the bibliography, the builder looks for its formatting information in the selected Reference Format table and if the type is not there, looks for the default values in the "%$types" table.

- The other entries in the top level table are keyed by Topic names and point to a structure containing all the data about a given Topic. This structure consists of pointers to three tables:
  - a table of abbreviations which consists of entries keyed by abbreviations pointing to full forms, e.g. "CJ" -> "Computer Journal";
  - a table of authors which contains an entry for each author name in the set of references of this topic - the entry points to a list of the references of which he is an author (this is used by the browser);

  and • a table of the references, which uses the citation key as a key for the table.

The Software Modules.

Our system has three main functions:

- The creation and maintenance of the database, including its super-structure and formatting information.
- The maintenance of the sets of references stored within the system.
- The process of building and outputting the bibliography to be attached to a paper.

The system has been constructed as a set of modules as follows:

The database maintenance modules include:

- A **creator** module which installs a skeletal database.
- Three **format loaders**, which pull in initial values for:
  - the set of **valid fields**;
  - the table of **Reference Types**;

  and • the table of **Reference Formats**.
- **Editors** for:
  - the set of **valid fields**;
  - the table of **Reference Types**;
  - the table of **Reference Formats**;
  - a **sort order**;
  - the set of **Topics**;

  and • a set of **abbreviations**.

The following modules are provided to maintain a set of references:

- A set of **bulk loaders**, which take a file of references in some format and insert them into the database. Thus the system can be initialised without the need for explicit data entry of all the items that the user has already. Bulk loaders for Scribe and Refer formats have already been written.

- An equivalent set of **dump programs**, so that databases can be moved from machine to machine and so that a hard copy of the complete library or the collection in a specific topic can be kept at hand whilst papers are being written. There is one dumper corresponding to each of the bulk loaders as well as one, "view", which provides output for viewing or for hard copy. This latter has access to the same Output Media as the bibliography builder.

- There is an **reference editor**. Its function is to insert and delete references and edit the values of their fields.

- Finally, a **browser** has also been supplied. At the moment, it only allows browsing by author name, although browsing by key-word will be a future development. The browser uses the table indexed on author name, described above.

The **bibliography builder** is in many ways the simplest part of the system. Its progress is through four stages:

- The paper is read in and all the "@cite"s are found and replaced by a bibliography key which is determined by the key layout for a reference of this Type.

- A list of all the citations is arranged in the specified sort order.

- When "@bibliography" or "@partbibliography" is encountered, the bibliography is formatted as described by the reference layout.

- The whole paper is finally translated to whichever output form is required, be it in a document production language like Postscript or a word processor like MacWrite, by being passed through one of the Output Media packages.

Finally, there are a number of low-level modules.

- A **Helper** handles the help information available;

- A **Chooser** creates menus from the identifiers of sets of objects. This is used to allow the selection of one of the valid field names, for instance. Quite often, these sets will be fairly large, so a combined

mouse and keyboard input mechanism has been provided. Initially, the program creates a menu with all of the item keys on it and shows the user 10 of them. Any item visible on the menu may be selected directly by clicking over the item with the mouse. If the required item is not visible, the menu may be scrolled in either direction by boxes which appear at the top and bottom of the menu. Alternatively, a character may be typed on the keyboard and this reduces the menu to those keys beginning with that character - subsequent key presses resulting in further reduction of the menu. Note that the menu is maintained dynamically so that it always reflects the current membership of the set of objects.

- A simple **String Editor** provides some basic facilities for the input and editing of a single line of text. The editor supports a cursor position in the string, scrolling to view parts of the string outside of the current window. The user may add new characters at the cursor position, delete the first character to the left of the cursor or delete all the characters the left of the cursor. The String Editor is shown in action at the bottom of Figures 4 and 8.

- A **form interface** allows forms to be presented on the screen consisting of boxed text strings or icons, which function as "light buttons". That is, when the mouse is used to select one of these buttons, an associated procedure is called. For instance, if a light button contains some text - the value of a field of a reference, say - selecting that button might call a text editor which allows the user to change the value of the field. This form interface, together with the PS-algol **menu** construct, has been used to construct the program as a hierarchy of procedures, simply chained together.

- A **More** facility displays textual information. The information is shown on the screen in a box. The output is paged in the same way as the UNIX "more" facility and the user clicks the mouse button to progress to the next page.

- An **error message** facility displays the message in a box at the centre of the screen, until the mouse button is pressed.

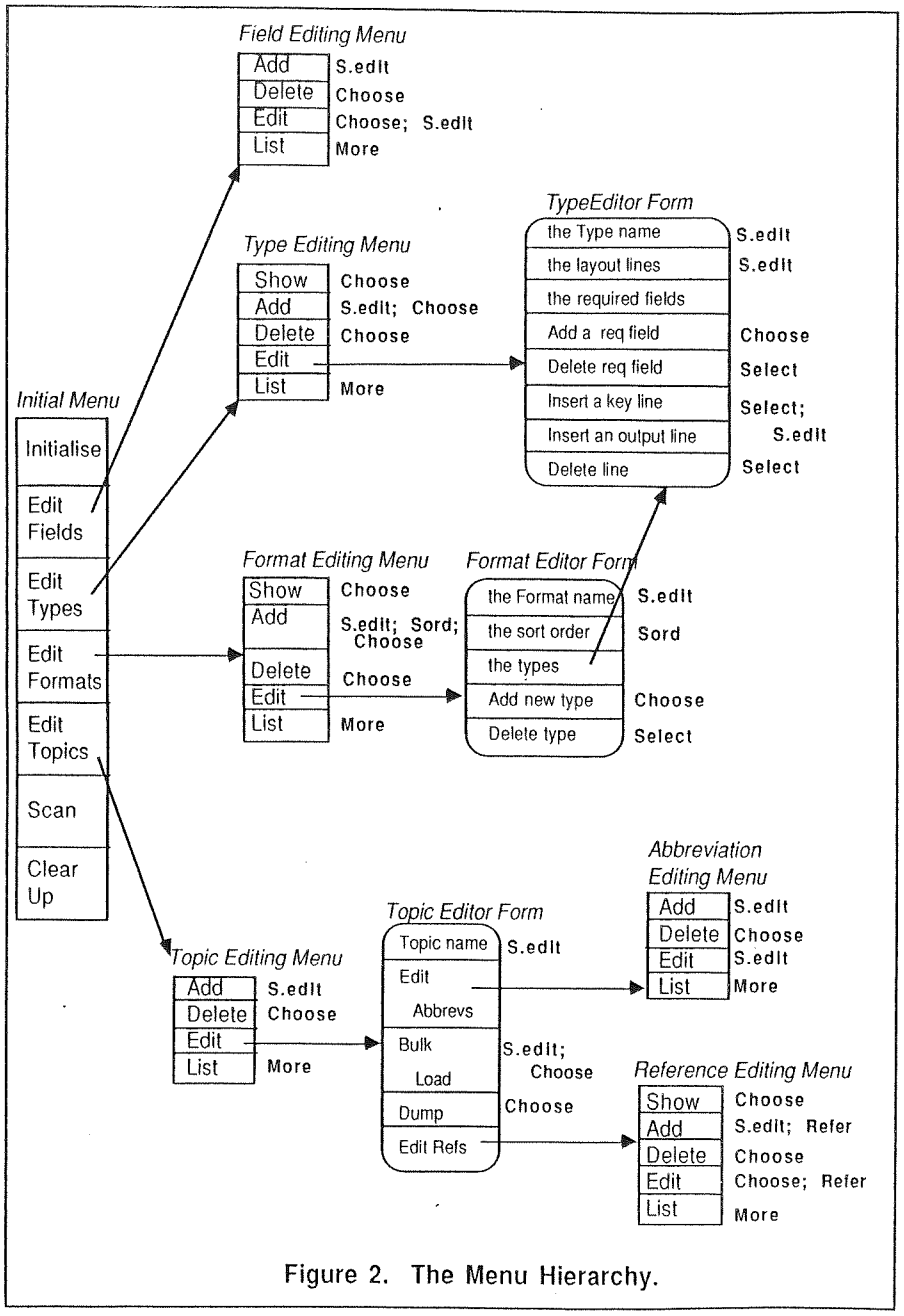

Figure 2. The Menu Hierarchy.

Introduction to Using the System.

The modules of the system are controlled through an interface consisting of a hierarchy of menus and forms, each of which contains options to obtain help and to quit to the next highest level. The other options either generate a further sub-menu or provide a dialogue which controls interaction with the user to achieve the operation selected.

The structure of the menu hierarchy is shown in Figure 2. Menus are shown in rectangular boxes and forms in rounded boxes. Moving down the hierarchy is achieved by clicking over a light button on one of the forms or menus. At the end of a chain of selections, the user interacts via a dialogue consisting of operations from the following list:

S.edit - a string is edited via the simple String Editor

Choose - an object of the required kind is selected from a menu of the identifiers of all the objects of that kind, via the **Chooser** module.

More - the requested text or list of object names is shown on the screen via the **More** module.

Select - the user indicates which object is to be operated on by clicking the mouse over the form element corresponding to that object.

Sord - call the Sort Order Editor (see Figure 6).

Refer - enter the Reference Editor (see Figure 8).

There are two important general points to be made about the way editing is performed by the user. Firstly, editing the identifier of an object causes the creation of a new object, with the old object being left unchanged. Secondly, each edit is structured as one of a set of nested transactions.

The major objects in the database have an identifying string associated with them. Fields, Types, Formats and Topics have names and the references themselves have a key. When the identifier of an object is changed by use of an editor, this corresponds to creating a new object. If the editor is entered with an object identified as "X", some changes are made to values within the object, then the identifier is changed to "Y" and then more changes are made, **a new object identified as "Y" will be created**. This will be a copy of "X" with all the modifications made, whether before or after modifying the identifier. This edit will leave "X" **totally unchanged**. Not even the changes made before the identifier will be made on "X". This method has been chosen so that many objects of the same type and with largely the same values can be created easily.

To illustrate the nested transaction mechanism, consider the process of editing one of the default set of Reference Types. To do this the user must do the following:

- Select the "Edit types" option in the top level menu. This initiates the transaction, "Edit the set of Types".
- Select the "Edit" option of the Type Editing Menu to edit a particular Type. This starts a sub-transaction, "Edit a Type".
- Edit the Type as described below.
- Respond to the question "Do you want to preserve your changes?". If the response is "y", the Type Editor returns a new Type object with the modified values. Otherwise, the editor returns the original object. Thus after the Type has been modified, the user can abandon the modifications at the end of the "Edit Type" transaction if so desired. If the modifications are kept, then they are held as part of the modifications in the current "Edit the set of Types" transaction.
- Edit more Types and when no more changes to the set of Types are to be made, respond to another "Do you want to preserve your changes?" question. This again gives the user the option of abandoning all changes done during the transaction by responding "n". If he responds "y" control returns to the initial menu and all the modifications are made to the database itself.
- To make the changes permanent, the user must select the Clear Up option and then select the "Commit" option, if he wishes then to carry on, or the "Quit/Commit" option, if he wishes to finish.

Getting Started.

The initial display and the first level of menus is shown as Figure 3. The start up screen consists of the heading and the vertical menu shown in bold on the left-hand side of the screen. The options of this menu have the following functions:

- **Help** - displays a short description of the options of this menu at the centre of the screen, until the mouse button is clicked - all "help" buttons function in this way;
- **Initialise** - sets up part or all of the database from scratch;
- **Edit fields** - allows the vector of field names to be edited;
- **Edit types** - allows the table of Reference Types to be edited;
- **Edit formats** - allows the tables of Reference Formats to be edited;

- **Edit topics** - allows the set of Topics to be edited;
- **Scan** - initiates the dialogue which leads to the building of a bibliography;
- **Clear Up** - handles both the commital of data to the database and exit from the program.

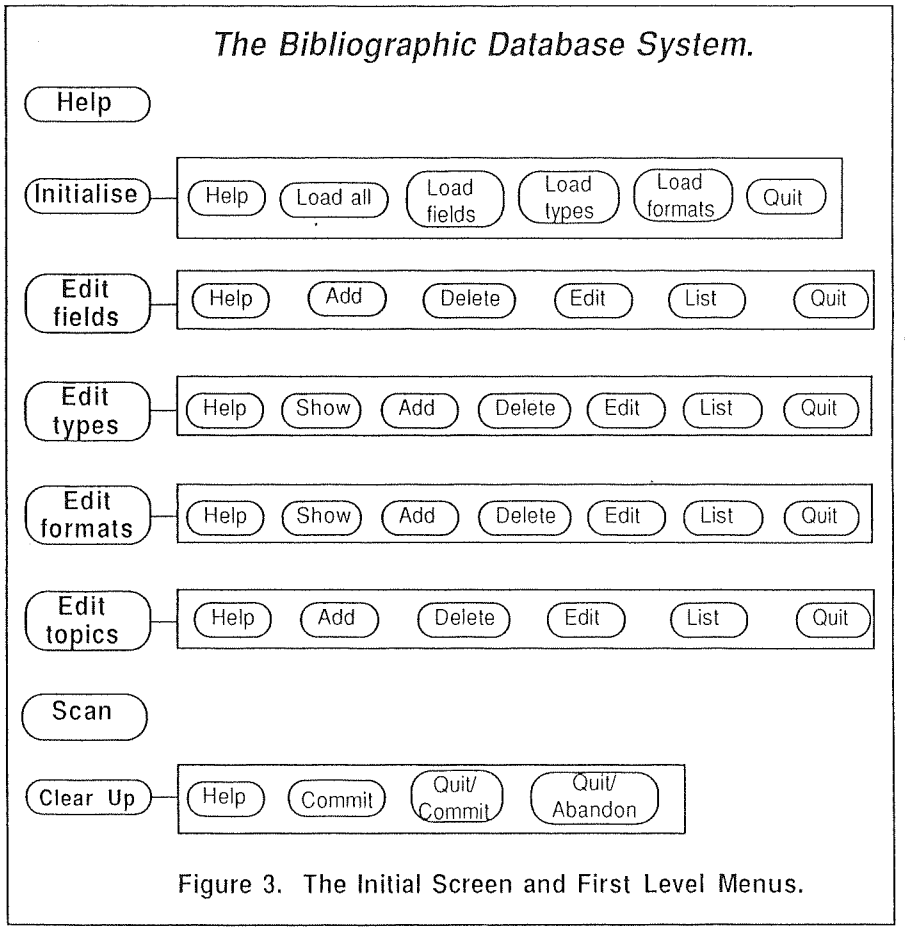

Figure 3. The Initial Screen and First Level Menus.

Initialising The Database.

Choosing the **Initialise** option of the initial menu brings up a sub-menu, which apart from the **help** and **quit** options common to all menus, has the following options:

Load fields - replace the valid field names vector with a list from the system file %SETUP;

Load types - replace the table of default Reference Types with a set found in %SETUP;

Load formats - replace the table of Reference Formats with those found in %SETUP;

Load all - replace all three of these from %SETUP.

The Set Editors.

Four of the options of the initial menu lead to sub-menus which control the editing of a set of objects. These sub-menus have similar structures. They all contain the options:

Add - add an item. Typically, this calls the String Editor to allow the user to specify an identifier for the new item and then makes further calls to the String Editor, to the Chooser or to the editor specific to an item of this kind to generate the values of other attributes of the item.

Delete - delete an item. The item to be deleted is selected via the Chooser.

Edit - edit the value of the item. Again an item is selected via the Chooser and then the current value is provided to the editor of the appropriate kind, which will announce itself by creating a new window in the screen to operate in. If the identifying information is edited (for instance, the Reference Type name), a new object is created and the old object is left intact. If only a change to the identifier is required, then after the editing has been done, the old object must be explicitly deleted.

List - provide a list of the identifiers of every item of this kind with the **More** module.

Additionally the menu for editing the Formats and the Types includes:

Show - display one of the items of this kind. The item to be displayed is selected by use of the Chooser. It remains on the screen until the mouse button is clicked.

Editing the Set of Known Field Names.

To edit the set of field names, choose the **Edit fields** option of the initial menu. The sub-menu contains the options, **Add**, **Delete**, **Edit** and **List**, which operate as just described. In particular:

- adding a field name consists of typing a new name into the String Editor;
- editing consists of choosing a field name and then changing it with the String Editor.

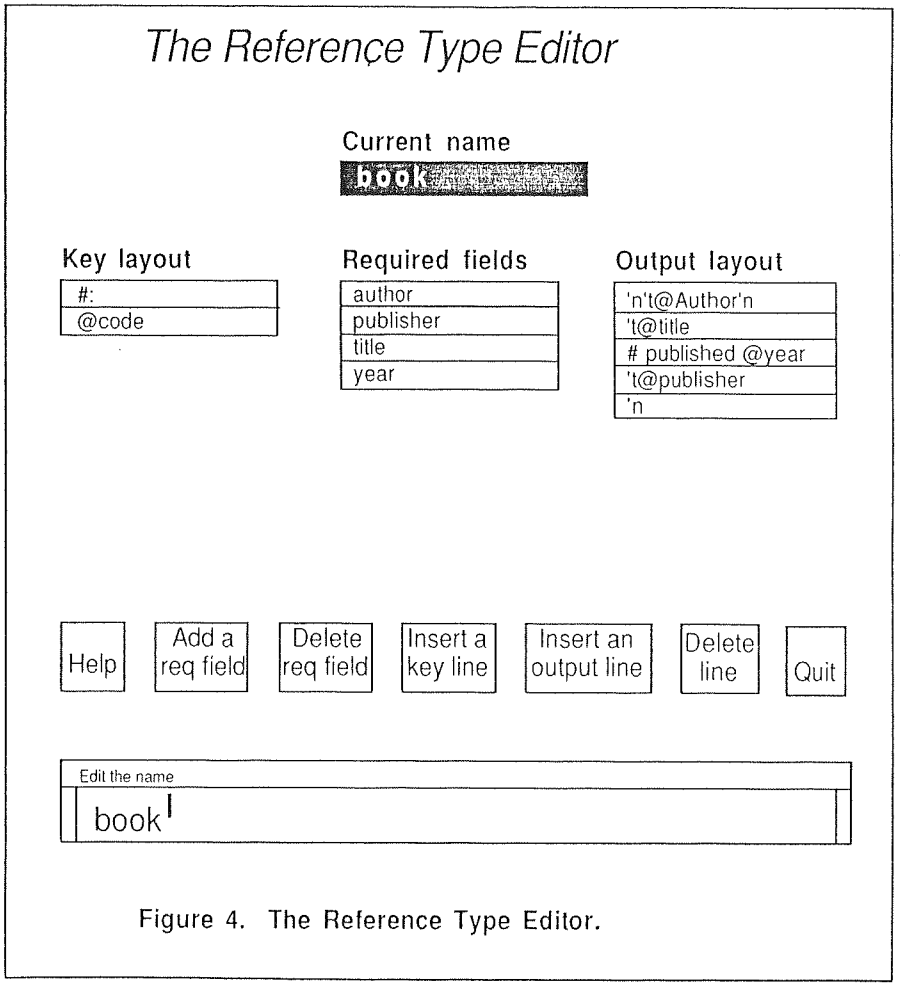

Figure 4. The Reference Type Editor.

Editing the Set of Default Reference Types.

Selecting the **Edit types** option of the initial menu summons the sub-menu, with all of the usual options, including **Show**, with the following particular details:

- Adding a new Type requires three calls to the String Editor to supply the Type name, a citation key layout and a reference layout. Then fields are added to the required fields list by menu selection from the vector of valid field names.

- Editing a Type requires the Type to be edited to be selected with the Chooser and then uses the Type Editor (Figure 4). This announces itself as a new window on the screen, within which the Type name is displayed at the top, under which is shown the information about the Type in three columns: one each for the key layout, output layout and the list of required field names. Clicking on the Type name or on any of the layout lines summons the String Editor to change them. The editor also has a row of light buttons at the bottom of the display, which include "help" and "quit" buttons and also:

  Add required field: select a field to add via the Chooser;

  Delete required field: click the mouse over the field to be deleted;

  Insert key layout line: click over the position at which the line is to be inserted and then input it via the String Editor;

  Insert output layout line: as for inserting a key layout line;

  Delete layout line: click the mouse over the line to delete.

- A Type is displayed by choosing the **show** option and then using the Chooser to pick which one to display. The Type is then displayed in a similar fashion to the layout of the editor.

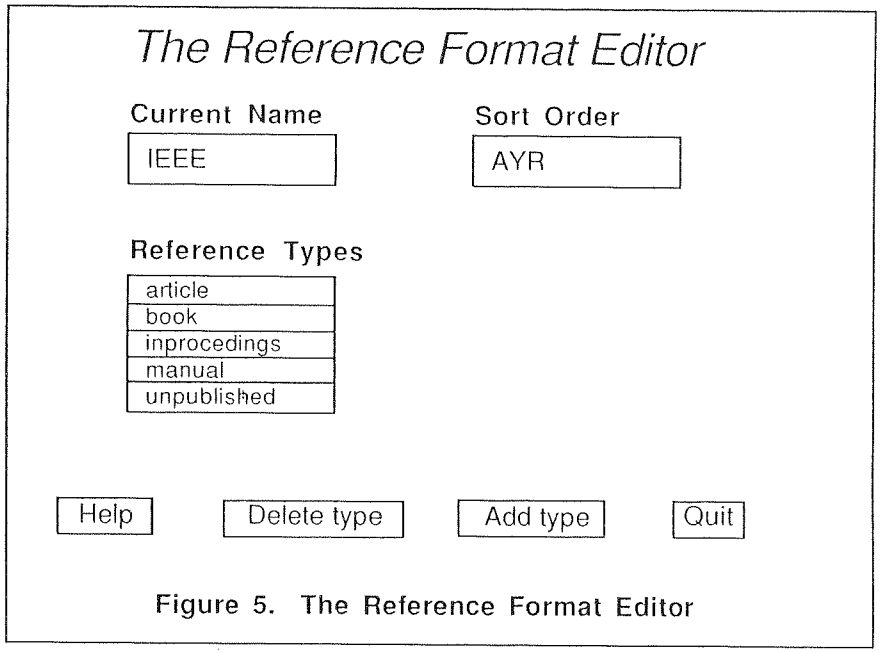

Figure 5. The Reference Format Editor

Editing the Reference Formats.

The **Edit formats** option of the initial menu brings up a sub-menu, which has the full set of five options which operate as already described, with the following particulars:

- Adding a new Reference Format consists of providing a new name via the String Editor and a sort order via the Sort Order Editor, which is described in the next section. Then the user loads in Reference Types from the default Reference Type table, via the Chooser.

- Editing a Reference Format, is done via the Reference Format Editor shown as Figure 5, after selection of a Format to edit by use of the Chooser. This displays the name and the sort order at the top of its window and the set of Reference Types vertically. Each of these may be clicked over to summon the String Editor, the Sort Order Editor or the Type Editor, respectively. There are further light buttons at the bottom of the display, including "help" and "quit" as usual, as well as buttons to add a new Type (via the Chooser) and delete a Type (by clicking the mouse over it).

- Displaying a Format requires selecting which one to show using the Chooser. It is then displayed in a layout similar to the editor's.

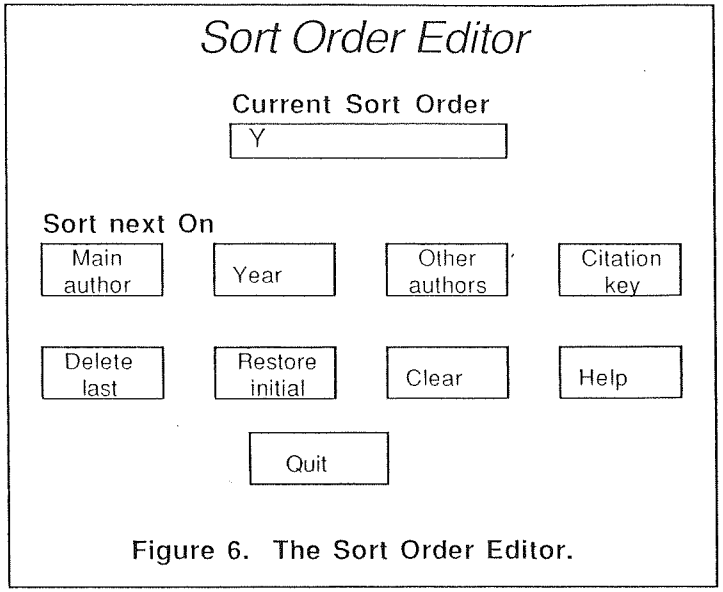

Figure 6. The Sort Order Editor.

Editing a Sort Order.

The sort order for a Reference Format is changed by using the Sort Order Editor shown in Figure 6. The current value of the sort order is displayed towards the top of the display, and under this there are nine light buttons, including the "Help" and "Quit" buttons. The buttons on the top line insert further sort key letters into the sort order string, thus adding fields to break ties between references which can be distinguished on the sort order so far. The other options gives the following operations:

- **Delete last** - remove the last sort key letter;
- **Restore initial** - return to the sort order string as it was on entry to the editor;
- **Clear** - clear the string to nothing.

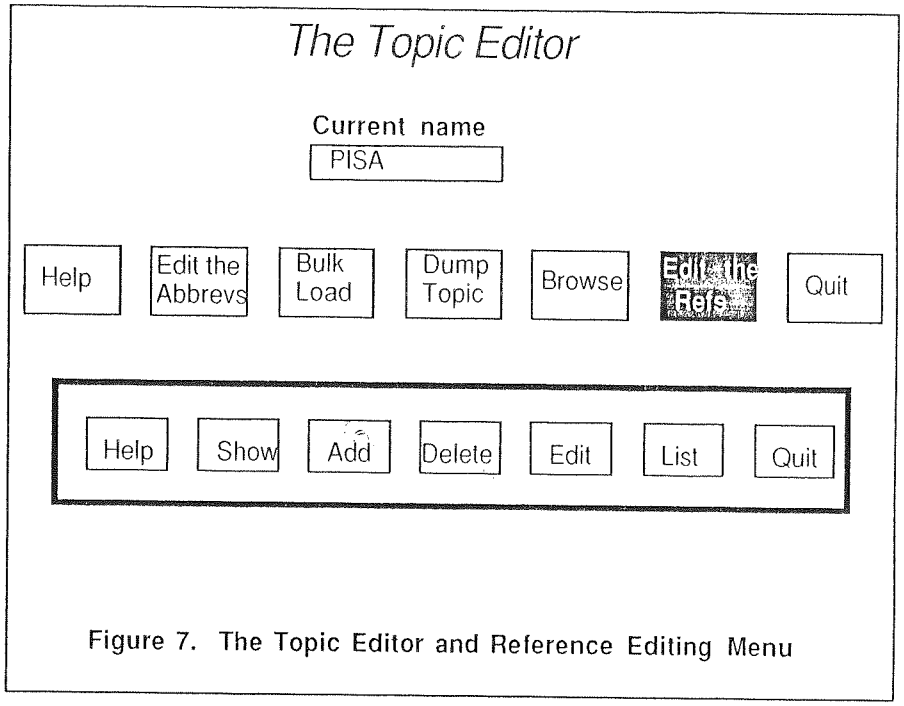

Figure 7. The Topic Editor and Reference Editing Menu

Editing The Topics.

The set of Topics may be edited by selecting the **Edit topics** option of the initial menu. The sub-menu which then appears doesn't include a "show" option, as there is too much information stored for each Topic to fit on the screen. The **Delete** and **List** options function as previously decribed, while:

Adding a new Topic requires a new name to be entered via the String Editor. A new entry in the database is created pointing to three empty tables which will hold the abbreviations, the author lists and the entries.

Selecting the edit option summons the Topic Editor which will is shown as Figure 7. It displays the name of the Topic at the top and this may be clicked on to call the String Editor to change it. Underneath this are a row of light buttons, including "Quit", "Help" and the following:

Bulk Load - a filename is requested using the String Editor and the format of the file is requested using the Chooser. All of the references and abbreviations found in the file are loaded into appropriate slots in the Topic's structure. At present, the file must be in Scribe format or Refer format.

Dump Topic - the contents of the Topic are dumped in a format selected by the Chooser from those available. If it is to be dumped in a reloadable format (e.g. Scribe), then a filename is requested using the String Editor. If, on the other hand, the dump is for viewing purposes, an Output Medium is selected via the Chooser. For further description of the Output Media, see the section on producing the bibliography.

Browse - the contents of the Topic are opened for browsing. The only browsing mechanism implemented as yet consists of traversing lists of papers with the same author. Therefore the browse option starts by requesting an author name by menu and then traversing the list by a menu of the following options:

List - display a list of all the keys;

Show - display details of the current paper;

Next - proceed to the next paper;

and **Find** - supply a year and go to the first paper of that year.

Edit the Abbrevs - a sub-menu appears with the usual structure for menus which control the editing of sets of objects. **Delete**, **List** and **Show** all function in the usual way. The other options work as follows:

adding an abbreviation requires two strings - the abbreviation and the full form - both of which are entered via the String Editor;

editing an abbreviation proceeds by selecting which to edit from a menu of the short forms and then modifying the short and long forms using the String Editor.

Edit the Refs - a sub-menu appears underneath the row of light buttons, which include the same set of options that have been seen in the higher level menus. The options **Delete** and **List** behave in the expected way, while **Show** displays an entry in a form compatible with the Reference Editor. The **Add** option requests a key via the String Editor and then calls the Reference Editor to fill in the fields. The **Edit** option calls the Chooser to select an entry to edit and then calls the Reference Editor.

The Reference Editor.

The Reference Editor is shown in Figure 8. At the top the following are displayed: the key under which it has been stored in the database; the Type of reference it is; and the list of authors. Underneath that field, are shown the required fields and under these, the optional fields. Clicking on the key or any of the fields results in the String Editor being called to modify these. Clicking on the Type allows it to be changed via the Chooser. Towards the bottom, there are a row of light buttons, including "Quit", "Help" and the following:

Add field - a new field name is selected from the set of valid field names and added to the list of optional fields. Then the String Editor is called to enter a value for the field.

Delete field - the field to be deleted is clicked over. Only optional fields can be deleted.

Abbrevs - clicking over this button throws a switch between displaying abbreviated strings in their short form (e.g. "@value[PPRR]) or their long form (e.g. "Persistent Programming Research Report").

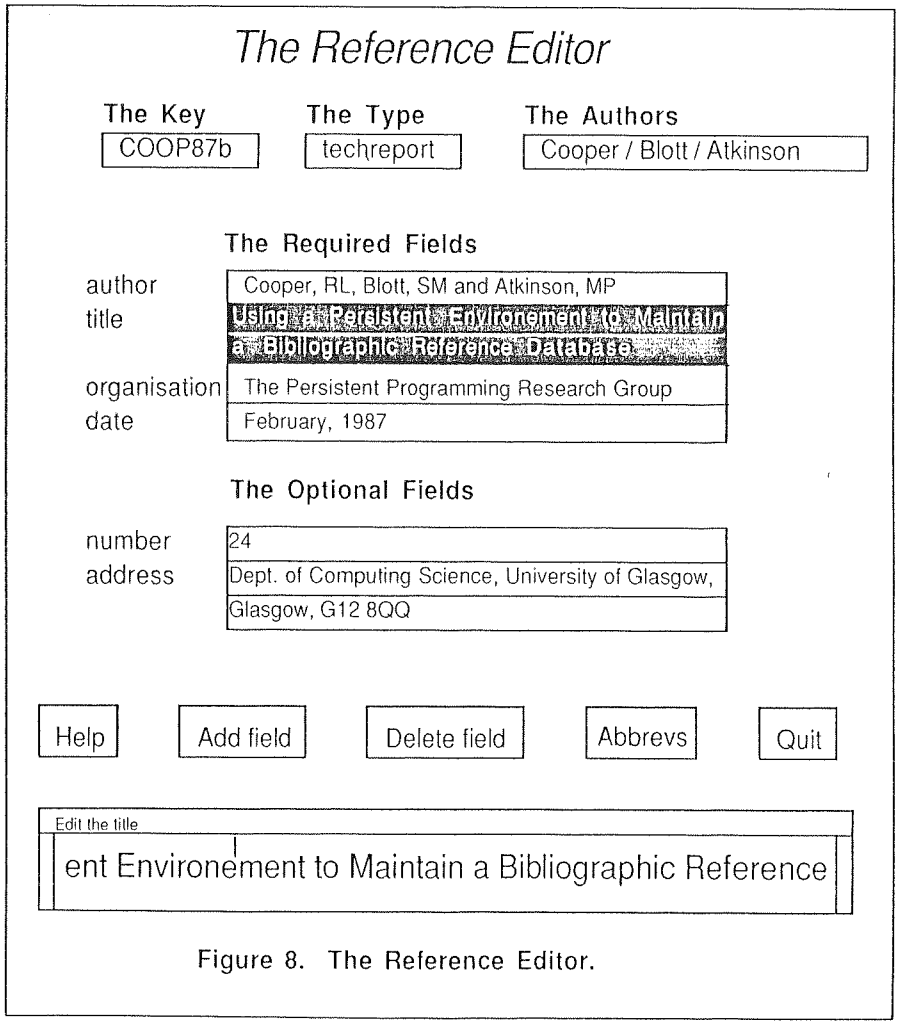

Figure 8. The Reference Editor.

Producing A Bibliography.

Having set up the database with all of the required information, using it to produce a bibliography proceeds as follows:

Create a text file containing the paper with all citations entered in the form "@cite[*ckey*]", where *ckey* is the citation key for the reference. The position of the bibliography should be indicated by a line containing just "@bibliography", to get all of the citations from the start to the current point, or "@partbibliography" to get all the citations from the last bibliography to the current point.

Enter the Bibliographic Database System and select "Scan" from the initial menu.

Supply, via the String Editor, a file name for the paper.

Choose a Reference Format from the menu provided.

Finally, supply an Output Medium, also by menu. This will be one of the following:

Screen - this option displays the output on the screen via More, that is paged with mouse button clicks to "turn" the page;

File - this sends the output to an ASCII file, the name of which is requested via the String Editor;

TEX - this sends the output to a file which formatted for input to a TEX processor.

Finishing Off.

The final option of the initial menu is labelled "Clear Up" and provides facilities for making the changes to the database permanent and for leaving the program. Selecting the option leads to a sub-menu, which includes a "Help" option as well as:

Commit - make any changes to the database permanent and continue within the system;

Quit/Commit - commit the changes and quit the system;

and **Quit/Abandon** - quit the system losing all the changes since the last commit.

Conclusions.

A system for the automatic creation of bibliographies has been described. All of the software was developed in a matter of four man-months. The speed of development was due to programming within a persistent environment. The system was developed in an incremental fashion, using fairly small, easily debugged modules. The modules were themselves stored in the same space as the data in the form of data structures containing properly bound first-class procedures. Therefore, it was easy to re-use sections of code to perform similar tasks. It was also easy to replace partially working modules with better ones.

However, the main benefit was the ability to store new parallel modules alongside old ones and then to generate menus to decide which module to use. For instance, initially the only bulk loader we had was for Scribe files. As soon as a Refer format file was encountered, we wrote a Refer format loader, plugged it in and allowed the Chooser to permit the selection dynamically via a menu.

All of the complex information required by the system is stored in a simple format. No recourse to file handlers or other complex schemes was required, other than to access the paper to be scanned and to produce the result. We think that the system would be more satisfactory if there were an option for the papers themselves to be stored in the persistent store. There is no technical reason why they should not be, but until word processing power has been added to the system, the disadvantages of having two copies of the paper probably outweigh the advantages.

Providing a page make-up system using the persistent store would give processing economy. Most runs of such a system are of iterations of the document, in which the document is only slightly perturbed. Retaining the data structures describing the layout of the generated pages would yield economies or an accelerated WYSIWYG response.

The use of the persistent store will continue to be exploited. We hope to have a group of from 10 to 30 people all accumulating their citation data in the one database, so that we economise on the effort of preparing this data. If possible, we will obtain and load more substantial reference information bodies. When these have been loaded, we can examine the performance and efficiency issues raised and utilise this as a demonstrator of persistence.

The larger accumulations of data will warrant better retrieval tools than our browser. We hope to build some on the basis of current information processing techniques, including some browsing on key words. The ability given by persistence to bind new code to existing, highly structured data, held in a strongly typed form should prove particularly helpful when adding such computationally sophisticated modules.

Bibliography.

APPL84 "The MacIntosh Manual", Apple Computers Inc., 1984.

ATKI85 Atkinson MP and Morrison R, "Procedures as First-class Objects", ACM TOPLAS 7, 4, 539-559 (Oct 1985).

ATKI86a Atkinson MP and Morrison R, "Integrated Persistent Programming Systems", proc 19th Annual Hawaii International Conference on Systems Sciences, Jan 7-10, 1986 (ed BD Shriver), vol IIA, Software, 842-854.

ATKI86b Atkinson MP, Morrison R and Pratten GD, "Designing A Persistent Information Space Architecture", *Proc Information Processing 1986*, North Holland Press (Sept 1986) 115 - 119.

ATKI87 Atkinson MP and Morrison R, "Binding and Type Checking in Database Programming Languages", in *Persistent Programming Research Report 34*, University of Glasgow, 1987.

CARD84 Cardelli L, "Amber", *Technical Report*, A T & T Bell Labs, Murray Hill, NJ, USA, 1984.

COOP87a Cooper RL, "Applications Programming in PS-algol", *Persistent Programming Research Report 25*, University of Glasgow, 1987.

COOP87b Cooper RL, Atkinson MP and Abderrahmane D, "Constructing Database Systems in a Persistent Environment", in *Persistent Programming Research Report 34*, University of Glasgow, 1987.

KNUT84 Knuth DE, "The TEX book", *Addison-Wesley Publishing Company*, 1984.

MORR86 Morrison R, Dearle A, Brown AL and Atkinson MP, "An Integrated Graphics Programming Environment", *Computer Graphics Forum*, Vol 5, No 2, June 1986, 147-157.

PPRR12 "The Ps-algol Reference Manual - Third Edition", *Persistent Programming Research Report 12*, University of Glasgow, 1987.

SCRB84 "The SCRIBE Document Production User Manual", UNILOGIC Ltd, 1984

Bibliography

Copies of documents in this list may be obtained by writing to:

The Secretary,
Persistent Programming Research Group,
Department of Computing Science,
University of Glasgow,
Glasgow G12 8QQ
Scotland.

Books

Davie, A.J.T. & Morrison, R.
"Recursive Descent Compiling", Ellis-Horwood Press (1981).

Atkinson, M.P. (ed.)
"Databases", Pergammon Infotech State of the Art Report, Series 9, No.8, January 1982. (535 pages).

Cole, A.J. & Morrison, R.
"An introduction to programming with S-algol", Cambridge University Press, Cambridge, England, 1982.

Stocker, P.M., Atkinson, M.P. & Grey, P.M.D. (eds.)
"Databases - Role and Structure", Cambridge University Press, Cambridge, England, 1984.

Published Papers

Morrison, R.
"A method of implementing procedure entry and exit in block structured high level languages". Software, Practice and Experience 7, 5 (July 1977), 535-537.

Morrison, R. & Podolski, Z.
"The Graffiti graphics system", Proc. of the DECUS conference, Bath (April 1978), 5-10.

Atkinson, M.P.
"A note on the application of differential files to computer aided design", ACM SIGDA newsletter Summer 1978.

Atkinson, M.P.
"Programming Languages and Databases", Proceedings of the 4th International Conference on Very Large Data Bases, Berlin, (Ed. S.P. Yao), IEEE, Sept. 78, 408-419. (A revised version of this is available from the University of Edinburgh Department of Computer Science (EUCS) as CSR-26-78).

Atkinson, M.P.
"Progress in documentation: Database management systems in library automation and information retrieval", Journal of Documentation Vol.35, No.1, March 1979, 49-91. Available as EUCS departmental report CSR-43-79.

Gunn, H.I.E. & Morrison, R.
"On the implementation of constants", Information Processing Letters 9, 1 (July 1979), 1-4.

Atkinson, M.P.
"Data management for interactive graphics", Proceedings of the Infotech State of the Art Conference, October 1979. Available as EUCS departmental report CSR-51-80.

Atkinson, M.P. (ed.)
"Data design", Infotech State of the Art Report, Series 7, No.4, May 1980.

Morrison, R.
"Low cost computer graphics for micro computers", Software Practice and Experience, 12, 1981, 767-776.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"PS-algol: An Algol with a Persistent Heap", ACM SIGPLAN Notices Vol.17, No. 7, (July 1981) 24-31. Also as EUCS Departmental Report CSR-94-81.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"Nepal - the New Edinburgh Persistent Algorithmic Language", in Database, Pergammon Infotech State of the Art Report, Series 9, No.8, 299-318 (January 1982) - also as EUCS Departmental Report CSR-90-81.

Morrison, R.
"S-algol: a simple algol", Computer Bulletin II/31 (March 1982).

Morrison, R.
"The string as a simple data type", Sigplan Notices, Vol.17,3, 46-52, 1982.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"Progress with Persistent Programming", presented at CREST course UEA, September 1982, revised in "Databases - Role and Structure", see PPRR-8-84.

Morrison, R.
"Towards simpler programming languages: S-algol", IUCC Bulletin 4, 3 (October 1982), 130-133.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"Problems with persistent programming languages", presented at the Workshop on programming languages and database systems, University of Pennsylvania. October 1982. Circulated (revised) in the Workshop proceedings 1983, see PPRR-2-83.

Atkinson, M.P.
"Data management", in Encyclopedia of Computer Science and Engineering 2nd Edition, Ralston & Meek (editors) January 1983. van Nostrand Reinhold.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"Algorithms for a Persistent Heap", Software Practice and Experience, Vol.13, No.3, 259-272 (March 1983). Also as EUCS Departmental Report CSR-109-82.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"CMS - A chunk management system", Software Practice and Experience, Vol.13, No.3 (March 1983), 273-285. Also as EUCS Departmental Report CSR-110-82.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"Current progress with persistent programming", presented at the DEC workshop on Programming Languages and Databases, Boston, April 1983.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"An approach to persistent programming", The Computer Journal, 1983, Vol.26, No.4, 360-365 - see PPRR-2-83.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"PS-algol a language for persistent programming", 10th Australian Computer Conference, Melbourne, Sept. 1983, 70-79 - see PPRR-2-83.

Morrison, R., Weatherill, M., Podolski, Z. & Bailey, P.J.
"High level language support for 3-dimension graphics", Eurographics Conference Zagreb, North Holand, 7-17, Sept. 1983. (ed. P.J.W. ten Hagen).

Cockshott, W.P., Atkinson, M.P., Chisholm, K.J., Bailey, P.J. & Morrison, R.
"POMS : a persistent object management system", Software Practice and Exerience, Vol.14, No.1, 49-71, January 1984.

Kulkarni, K.G. & Atkinson, M.P.
"Experimenting with the Functional Data Model", in Databases - Role and Structure, Cambridge University Press, Cambridge, England, 1984.

Atkinson, M.P. & Morrison, R.
"Persistent First Class Procedures are Enough", Foundations of Software Technology and Theoretical Computer Science (ed. M. Joseph & R. Shyamasundar) Lecture Notes in Computer Science 181, Springer Verlag, Berlin (1984).

Atkinson, M.P., Bocca, J.B., Elsey, T.J., Fiddian, N.J., Flower, M., Gray, P.M.D. Gray, W.A., Hepp, P.E., Johnson, R.G., Milne, W., Norrie, M.C., Omololu, A.O., Oxborrow, E.A., Shave, M.J.R., Smith, A.M., Stocker, P.M. & Walker, J.
"The Proteus distributed database system", proceedings of the third British National Conference on Databases, (ed. J. Longstaff), BCS Workshop Series, Cambridge University Press, Cambridge, England, (July 1984).

Atkinson, M.P. & Morrison, R.
"Procedures as persistent data objects", ACM TOPLAS 7, 4, 539-559, (Oct. 1985) - see PPRR-9-84.

Morrison, R.,Bailey, P.J., Dearle, A., Brown, P. & Atkinson, M.P.
"The persistent store as an enabling technology for integrated support environments", 8th International Conference on Software Engineering, Imperial College, London (August 1985), 166-172 - see PPRR-15-85.

Atkinson, M.P. & Morrison, R.
"Types, bindings and parameters in a persistent environment", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 1-24 - see PPRR-16-85.

Davie, A.J.T.
"Conditional declarations and pattern matching", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 278-283 - see PPRR-16-85.

Krablin, G.L.
"Building flexible multilevel transactions in a distributed persistent environment, proceedings of Data Types and Persistence Workshop, Appin, August 1985, 86-117 - see PPRR-16-85.

Buneman, O.P.
"Data types for data base programming", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 291-303 - see PPRR-16-85.

Cockshott, W.P.
"Addressing mechanisms and persistent programming", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 363-383 - see PPRR-16-85.

Norrie, M.C.
"PS-algol: A user perspective", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 399-410 - see PPRR-16-85.

Owoso, G.O.
"On the need for a Flexible Type System in Persistent Programming Languages", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 423-438 - see PPRR-16-85.

Morrison, R., Brown, A.L., Bailey, P.J., Davie, A.J.T. & Dearle, A.
"A persistent graphics facility for the ICL PERQ", Software Practice and Experience, Vol.14, No.3, (1986) - see PPRR-10-84.

Atkinson, M.P. and Morrison R.
"Integrated Persistent Programming Systems", proceedings of the 19th Annual Hawaii International Conference on System Sciences, January 7-10, 1986 (ed. B. D. Shriver), vol IIA, Software, 842-854, Western Periodicals Co., 1300 Rayman St., North Hollywood, Calif. 91605, USA - see PPRR-19-85.

Atkinson, M.P., Morrison, R. and Pratten, G.D.
"A Persistent Information Space Architecture", proceedings of the 9th Australian Computing Science Conference, January, 1986 - see PPRR-21-85.

Kulkarni, K.G. & Atkinson, M.P.
"EFDM : Extended Functional Data Model", The Computer Journal, Vol.29, No.1, (1986) 38-45.

Buneman, O.P. & Atkinson, M.P.
"Inheritance and Persistence in Database Programming Languages"; proceedings ACM SIGMOD Conference 1986, Washington, USA May 1986 - see PPRR-22-86.

Morrison R., Dearle, A., Brown, A. & Atkinson M.P.; "An integrated graphics programming environment", Computer Graphics Forum, Vol. 5, No. 2, June 1986, 147-157 - see PPRR-14-86.

Atkinson, M.G., Morrison, R. & Pratten G.D.
"Designing a Persistent Information Space Architecture", proceedings of Information Processing 1986, Dublin, September 1986, (ed. H.J. Kugler), 115-119, North Holland Press.

Brown, A.L. & Dearle, A.
"Implementation Issuses in Persistent Graphics"; The Association for Computing Machinery, 11 West 42nd St., New York, NY 10036; University Computing, Vol. 8, N0. 2, (Summer 1986) - see PPRR-23-86.

Internal Reports

Morrison, R.
"S-Algol language reference manual", University of St Andrews CS-79-1, 1979.

Bailey, P.J., Maritz, P. & Morrison, R.
"The S-algol abstract machine", University of St Andrews CS-80-2, 1980.

Atkinson, M.P., Hepp, P.E., Ivanov, H., McDuff, A., Proctor, R. & Wilson, A.G.
"EDQUSE reference manual", Department of Computer Science, University of Edinburgh, September 1981.

Hepp, P.E. and Norrie, M.C.
"RAQUEL: User Manual", Department of Computer Science Report CSR-188-85, University of Edinburgh.

Norrie, M.C.
"The Edinburgh Node of the Proteus Distributed Database System", Department of Computer Science Report CSR-191-85, University of Edinburgh.

In Preparation

Atkinson, M.P. & Buneman, O.P.
"Database programming languages design", submitted to ACM Computing Surveys - see PPRR-17-85.

Theses

The following Ph.D. theses have been produced by members of the group and are available from the address already given,

W.P. Cockshott
Orthogonal Persistence, University of Edinburgh, February 1983.

K.G. Kulkarni
Evaluation of Functional Data Models for Database Design and Use, University of Edinburgh, 1983.

P.E. Hepp
A DBS Architecture Supporting Coexisting Query Languages and Data Models, University of Edinburgh, 1983.

G.D.M. Ross
Virtual Files: A Framework for Experimental Design, University of Edinburgh, 1983.

G.O. Owoso
Data Description and Manipulation in Persistent Programming Languages, University of Edinburgh, 1984.

Persistent Programming Research Reports

This series was started in May 1983. The following list gives those produced and those planned plus their status at 10th April 1987. Copies of documents in this list may be obtained by writing to the address already given.

PPRR-1-83 The Persistent Object Management System - Atkinson, M.P., Chisholm, K.J. and Cockshott, W.P. £1.00

PPRR-2-83 PS-algol Papers: a collection of related papers on PS-algol - Atkinson, M.P., Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R. £2.00

PPRR-4-83 The PS-algol reference manual - Atkinson, M.P., Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R Presently no longer available

PPRR-5-83 Experimenting with the Functional Data Model - Atkinson, M.P. and Kulkarni, K.G. £1.00

PPRR-6-83 A DBS Architecture supporting coexisting user interfaces: Description and Examples - Hepp, P.E. £1.00

PPRR-7-83 EFDM - User Manual - K.G.Kulkarni £1.00

PPRR-8-84 Progress with Persistent Programming - Atkinson, M.P., Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R. £2.00

PPRR-9-84 Procedures as Persistent Data Objects - Atkinson, M.P.,Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R. £1.00

PPRR-10-84 A Persistent Graphics Facility for the ICL PERQ - Morrison, R., Brown, A.L., Bailey, P.J., Davie, A.J.T. and Dearle, A. £1.00

PPRR-11-85 PS-algol Abstract Machine Manual £1.00

PPRR-12-86 PS-algol Reference Manual - third edition £2.00

PPRR-13-85 CPOMS - A Revised Version of The Persistent Object Management System in C - Brown, A.L. and Cockshott, W.P. £2.00

PPRR-14-86 An Integrated Graphics Programming Environment - second edition - Morrison, R., Brown, A.L., Dearle, A. and Atkinson, M.P. £1.00

PPRR-15-85 The Persistent Store as an Enabling Technology for an Integrated Project Support Environment - Morrison, R., Dearle, A, Bailey, P.J., Brown, A.L. and Atkinson, M.P. £1.00

PPRR-16-85 Proceedings of the Persistence and Data Types Workshop, Appin, August 1985 - ed. Atkinson, M.P., Buneman, O.P. and Morrison, R. £15.00

PPRR-17-85 Database Programming Language Design - Atkinson, M.P. and Buneman, O.P. £3.00

PPRR-18-85 The Persistent Store Machine - Cockshott, W.P. £2.00

PPRR-19-85 Integrated Persistent Programming Systems - Atkinson, M.P. and Morrison, R. £1.00

PPRR-20-85 Building a Microcomputer with Associative Virtual Memory - Cockshott, W.P. £1.00

PPRR-21-85 A Persistent Information Space Architecture - Atkinson, M.P., Morrison, R. and Pratten, G.D. £1.00

PPRR-22-86 Inheritance and Persistence in Database Programming Languages - Buneman, O.P. and Atkinson, M.P. £1.00

PPRR-23-86 Implementation Issues in Persistent Graphics - Brown, A.L. and Dearle, A. £1.00

PPRR-24-87 Using a Persistent Environment to Maintain a Bibliographic Database - Cooper, R.L., Atkinson, M.P. & Blott, S.M. £1.00

PPRR-25-87 Applications Programming in PS-algol - Cooper, R.L. £1.00

PPRR-26-86 Exception Handling in a Persistent Programming Language - Philbrow, P & Atkinson M.P. £1.00

PPRR-27-86 A Context Sensitive Addressing Model - Hurst, A.J. £1.00

PPRR-28-86b A Domain Theoretic Approach to Higher-Order Relations - Buneman, O.P. & Ochari, A. £1.00

PPRR-29-86	A Persistent Store Garbage Collector with Statistical Facilities - Campin, J. & Atkinson, M.P.	£1.00
PPRR-30-86	Data Types for Data Base Programming - Buneman, O.P.	£1.00
PPRR-31-86	An Introduction to PS-algol Programming - Carrick, R., Cole, A.J. and Morrison, R.	£1.00
PPRR-34-87	Binding Issues in Database Programming - Atkinson, M.P., Morrison, R., Cooper, R.L. and Abderrahmane, D.	£1.00