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 9SS

Andrew, Unix and Educational Computing

Wilfred J. Hansen

Persistent Programming
Research Report 40
June 1987

Alan Deale.

Preface

This report contains two papers by Professor Wilfred J. Hansen:

Andrew and Unix

The Andrew Environment for Development of Educational Computing

Andrew and Unix

Wilfred J. Hansen

Computing Science Department
University of Glasgow
Glasgow, G12 8QQ
Scotland

Information Technology Center
Carnegie-Mellon University
Pittsburgh, PA 15213
USA

December, 1986

This note sketches some of the ways in which choice of Unix as an operating system influenced the design of Andrew, the workstation software developed by the Information Technology Center at Carnegie-Mellon University as a joint effort between the University and IBM [Morris, 1986]. (UNIX is a trademark of AT&T Bell Laboratories.) I will consider the file system, window manager, menus, text manager, and scrollbars.

The target of the ITC effort was to produce a system so students faculty and staff could access a uniform file system from any of 5000 or more personal workstations of the IBM 6150 class (RT/PC). These workstations provide computing power sufficient for students to utilise application packages similar or identical to those they will encounter in industry. A shared file system was important because it provides communication between users without the physical transport of floppy disks; the campus had grown accustomed to a shared file system from years of time-sharing.

Unix Features

Three aspects of Unix 4.2BSD had a significant impact on our planning of the Andrew: sockets, select, and its non-real-time nature.

Sockets provide a convenient interprocess communication mechanism, especially when used in a client-server model. A server process listens on a socket bound to a well known port and clients connect to it by referring to the same port. As a result of the connection sequence, each process possesses a file descriptor on which it can read and write to communicate with its opposite. We have usually used sockets in a remote procedure call mode: applications in the client process call procedures which reside in the server. A specialised interface intercepts the call, passes the parameters to the server, awaits the response, and passes any returned value back to the client.

Select enables a process to await input on more than one file descriptor. This is crucial because a server may not have any way to predict which client will next submit input. For example, several applications serve as monitors executing other applications. They wait at a select for input to arrive either from the process running the application or from the keyboard or mouse.

Real-time constraints are difficult to meet in Unix because processes share the processor and none is guaranteed any given level of service. For this reason we initially designed the user interface to make minimal use of dynamic interaction techniques such as dragging with the mouse. As we have progressed to faster hardware, we have increasingly turned toward more dynamic interaction.

File System

The most important impact of Unix on Vice, the Andrew file system, is in the design of the semantics. The entire campus-wide file system is made to appear as a single Unix file tree, so the name of any given file depends on its position in the hierarchy rather than its physical location. The Unix protection scheme is observed, but has been augmented by a more powerful scheme which permits precise specification of which users can access the files in any directory.

To provide universal access to files, they are stored in file servers accessible from all workstations. When a file is opened, it is transferred from the server to a cache directory on the workstation; thereafter, access to the file is performed on the local copy. If the file is modified, it is copied back to the server when the file is closed. For load balancing, the files of any given user may be moved from one server to another, and this is transparent to the user.

The file server utilises *select* to await various forms of events, and communicates with clients through a socket. In this case one socket communicates with all clients because there are not enough file

descriptors to give one to each client. Instead, the remote procedure call implementation sends packets directly, performing itself all the functions required for message transport, including checkpointing, fragmentation, sequencing, and error recovery. A special purpose light-weight process system allows one server process handles requests from all clients.

Within client workstations a user-level process called Venus handles communication with the file server. There are a few kernel modifications which trap file opens, closes, and other crucial system calls. These traps transmit messages through a file descriptor to Venus which then communicates to the file server, updates the local file cache, and returns status information to the kernel. At the heart of Venus is a select waiting for messages from the kernel and file server.

Window Manager

The Andrew window manager allocates the screen space under user control, draws images thereon for clients, and multiplexes the mouse and keyboard to send inputs to the proper client.

The usual initial reaction to Andrew windows is to notice that they are tiled on the screen rather than overlapping. Initially this was done in the belief that tiled windows could be more quickly implemented and that overlapping windows could later be added in a manner transparent to the application. By the time we understood that the latter hope was incorrect we had found that we liked tiled windows because they avoid "window washing," a phenomenon where users spend considerable non-productive time managing screen allocation manually.

(The implementation problem with overlapping windows lies in the redraw strategy. In Andrew, a client is told simply that the window contents must be redrawn, but for overlapping windows it is important to be able to tell a client which portion of a window needs to be redrawn. To use overlapping windows Andrew client programs can be revised to accept and act on partial redraw requests.)

A variety of mechanisms have been chosen for the relationship between window manager and client applications [NeWS, 1986]. For Andrew the window manager was made a separate process that contains the display screen bitmap in its process space. This choice was taken partly because 4.2BSD Unix does not offer shared memory and partly because we wished to avoid the kernel modifications that would be necessary to protect the screen space of clients one from another.

As with the file system, the heart of the window manager-client

interface is sockets and select. The remote procedure call mechanism in the client queues up messages to the window manager requesting it to draw lines, perform raster-ops, fill trapezoids, and display text. When these messages are sent across the socket, the select call in the window manager indicates which client made the request so the image operations can be carried out on the corresponding window. One reason the whole process is not intolerably inefficient is that very few of the procedure calls to the window manager return results. Thus many messages are queued together while the client process executes and all are performed together after the messages are transmitted and the window manager is swapped in.

Implementing the window manager as a separate, user-level process has made it easy to port to a variety of workstations and display devices. It has also made it trivial to run applications on one processor and view the results through windows on a workstation. For example, we have a large Unix mainframe without bitmapped displays of its own; to access it we run the Andrew shell interface and text editor on it from a bitmapped workstation, getting the combination of a mainframe and an interactive interface.

Menus

Menus in Andrew are "attached" to the middle mouse button. When it is pressed, the menu pops up at the current position of the cursor. The menu appears as a stack of cards, with those behind the first staggered to the northwest, Figure 1. As the mouse is moved to the left, cards behind the first are made visible, obscuring those in front, Figure 2; and as the cursor is moved to the right cards more forward in the stack reappear. In general, the visible card is the one for which the cursor is in the left margin. As the cursor is moved up or down the margin, the menu option on that line is highlighted in reverse video. If the mouse button is released while there is a highlighted option, it will be executed.

The non-realtime nature of Unix contributed to the decision to include the menu package as part of the window manager. Mouse tracking information is transmitted directly to the window manager and acted on there without a process swap to the client. This also gives the advantage that the menu package is the same for all applications and can be changed consistently without rebuilding the application code.

Many interactive systems have menus that reside in some particular place on the screen. There are two problems with this scheme, depending on how often the given menu option is chosen. If the option is chosen frequently, the user must move each time to the position on the screen containing the option; this can require major arm movement on large

screens. If the option is chosen infrequently, then the screen space devoted to it can probably be better spent on some more useful information. With the Andrew pop-up menus, 30 or more options can be available at any point on the screen and large numbers of infrequent options can be made available by providing a pop-up menu from a screen space. For example, the small icon at the right of windows titles in Figure 1 refers to a menu with 20 infrequently chosen options.

Text Manager

Text is managed for Andrew applications by a subroutine package that offers full typographic quality text: various fonts and point sizes, margins, tabs, justification, and so on. Text on the screen is generally justified to both the left and right edges of the space available. (This is not WYSIWYG; the text is justified to the screen space it occupies rather than to the arbitrary limits of a piece of paper.) Among the applications that employ the text manager are a text editor, a shell interface, and mail tools.

Unix influenced the design of the text manager rather peripherally. The fact that 4.2BSD implements virtual memory as well as it does made it possible to utilise an internal data structure where the text of the document is stored as a single sequential string [Hansen, 1987]. Printing of text is implemented by converting from the internal format to troff input.

Scrollbar

Part of the text manager package is a scrollbar that shows the user what part of the file is visible and offers options to view any portion of the file. The scrollbar appears to the left of the text it controls, as shown in Figure 1. The relation of the white portion of the scrollbar to the entire scrollbar is the same as the relation of the visible portion of the file to the entire file. The white portion may be "dragged" with the mouse so any random portion of the file can be made visible. To scroll forward in the file by a short distance, the user presses the left button opposite some line of the visible text, which is then moved to the top of the window. This provides for scrolling forward by an arbitrary distance up to the height of the window. Similarly, the right button scrolls backward.

Since Unix is not a realtime system, we initially did not implement the scrollbar with dragging. Pressing on the left half of the scrollbar scrolled up or down as described for clicking above, and pressing on the right half scrolled to the corresponding arbitrary point in the file. A few experiments showed that dragging was possible even with the

process swap from window manager to client and back, so the current scrollbar with dragging was implemented. Even then, the amount of change in the image for any movement of the mouse is small: a few rows of white and a few rows of grey are redrawn. To limit the possible interaction even further, the mouse cursor is constrained to remain within the scrollbar while the button is down.

[Hansen, 1987] Hansen, Wilfred J., *Data Structures for a Bit-Mapped Text Editor*, Byte (Jan 1987, scheduled).

[Morris, 1986] Morris, J.H., et al., *Andrew: A Distributed Personal Computing Environment*, Comm ACM 29, 3 (Mar 1986) 184-201.

[NeWS, 1986] Sun Microsystems, **NeWS Preliminary Technical Overview** (Oct 1986).

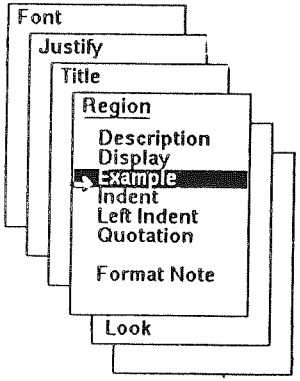

Figure 2. Popping through the menu. This is the same menu as in Figure 1, b ut the cursor has been moved to the left.

Figure 1. A view of the Andrew screen. Applications shown include the clock, the shell interface, the text editor, and the editor for text styles. The menu is the one made available inside selected text; it provides the options for giving typographic styles to text.

The Andrew Environment for Development of Educational Computing

Wilfred J. Hansen

Information technology Center
Carnegie-Mellon University

and

Computing Science Department
University of Glasgow

April, 1987

Abstract: Andrew is being developed at Carnegie-Mellon University as an environment for computing by all members of the campus community. The system includes a file system, user interface software, and the CMU-Tutor facility for building instructional material. This demonstration focuses on the latter environment, showing how it provides excellent facilities for generating images and evaluating student responses to questions.

I. Introduction

Carnegie-Mellon University recognized some years ago that computing is an important community resource. They began, shortly thereafter, joint development with IBM of the system which has since become known as Andrew [Morris 1986]. The goal of this effort is to make possible universal interconnection of advanced workstations, one for each student, staff, and faculty member. These computers support training by offering the tools that professionals use in practice. They support research by providing common tools of statistical analysis and data bases. Perhaps even more importantly, they support the community with the mundane tasks of communication: electronic mail, class work, scholarly publications, and documents of all kinds.

With such pervasity of computing, it has become possible to explore non-traditional education with computers. In pursuit of this, the Tutor system was seen as the most promising of the approaches and a similar approach was developed for Andrew. This is the CMU-Tutor programming environment for the construction of "lessons," where each lesson is a tutorial program which leads a student through learning some topic of knowledge [Sherwood, 1986a; 1986b].

In this paper I will briefly sketch the Andrew file system and user interface and then describe the CMU-Tutor environment.

II. Vice, The Andrew File System

Vice, the Andrew file system, enables each user to sit at any workstation on campus and access all files to which he or she has access from any workstation. Moreover, this is done within a name hierarchy that hides for the user the physical location of his or her files. The user's view of a file is a name, a set of access permissions, and a contents. The files for user xxx in department ddd will all have names beginning with /cmu/ddd/xxx/, a top-down hierarchical structure. The access permission scheme is more detailed than that of other systems: each directory may be controlled as to exactly which users or groups of users have access to it.

The file system stores files in dedicated file server computers. These and all other workstations on campus are connected via IBM token ring technology. Each workstation has a local disk to cache files and a *Venus* process to communicate with the file server. When any other process on the workstation tries to open a file, Venus is notified, communicates with the server to get the file to the cache and supplies the cached copy of the file to the process.

III. Virtue, The Andrew User Interface

Andrew is predicated upon advanced workstations with large screen, bit-mapped displays, so it was necessary to develop user interface software to provide for sharing the screen among applications and exploiting the graphic and textual potential. The lowest level of this software is a window manager, *wm*, which tiles the screen into a separate window for each application, as shown in Figure 1. In this Figure, the window in the upper left is a clock, the window in the lower left is a shell interface for giving commands to the system, the window in the upper left is a text editor, and the window in the lower left is the educational lesson *gt*.

The *wm* window manager is implemented as a separate process to which other processes communicate for window allocation and graphic services. It also listens to the keyboard and mouse, distributing inputs from them to the appropriate destination processes. The *gt* window, for example, draws the axes of its graph with calls like these:

```
wm_MoveTo(GX(0), GY(500));
wm_DrawTo(GX(0), GY(0));
wm_DrawTo(GX(25), GY(0));
```

where GX and GY are macros which convert from coordinates for the graph to coordinates within the *gt* window.

The three windows in Figure 1 other than the clock window all employ Andrew's *base editor* for manipulation of text. This facility provides full typographic text including fonts, font sizes, indentation, justification, tabs, and sub- and superscripts. User manipulation of this text is the same in all windows so the user need learn only one set of conventions for dealing with text. The advantages for the programmer are enormous because individual programs need not handle text interaction.

The *gt* application is a prototypical educational application, and was the first constructed for Andrew. It is a simulation intended to teach the relationships of position, velocity, and acceleration. The graph in the middle of the window depicts six posts with ramps between them along which a ball can roll. The height of each post can be adjusted by pointing above or below its top and clicking the left mouse button. Below the posts and ramps are two bars to select the initial position and velocity for the ball; the user chooses a value from these bars by pointing at a position on them and clicking the mouse. When the user selects the Roll Ball option from the menu, the ball rolls on the ramps and a graph of its position, velocity, or acceleration is displayed in the top portion of

the window. To give the user a goal, a graph is shown for a selected configuration of the parameters and the user is challenged to match that graph.

Gt illustrates two important differences between educational lessons on bit-mapped workstations with mice and small screen personal computers without. First, the graphics are designed to adapt to any size window the user may choose.* This is essential anyway because it cannot be known what display hardware will be available. Second, there is no "dialogue" between system and user to adjust the apparatus. With PC based versions of *gt*, the user is forced to answer a series of questions: *Do you want to adjust the parameters?* yes *Do you want to change a post height?* yes *Which post?* 4 *Height is 1. What new height?* 2 *Do you want to change another post? no Do you want to change the initial velocity?* no *Do you want ad nauseum . . .* This entire exchange in *gt* is a simple click above the second post. After then choosing two menu options the image looks as in Figure 2. Both this simple command interface and the adaptability of size contribute to a feeling in the user that he or she, rather than the computer, is in control.

(footnote)
* Perhaps a short digression on variable-sized windows is appropriate. The small additional programming complexity for this adaptability amounted to about a page of code in *gt*, but helps the system build in the user confidence that he or she is in control rather than the computer. Programmers who have not yet written Andrew applications sometimes complain that this makes rational planning of screen usage difficult, but after experiencing use of the system in developing material, such complaints vanish. If the user gives an application a window that is too small, the application may choose to complain and wait for a larger window; but if it just goes ahead and does its best a dissatisfied user can easily adjust the window size.

IV. CMU-Tutor

Broadly defined, educational computing is any use of a computer that teaches someone something. Andrew supports all such computing. Sometimes, however, it is desirable to use a computer for presentation of material, as could be done with a book. The advantage of the computer is that it can offer animations and other interaction with the student. The pace of the presentation can be controlled by the author and the student can be questioned to ensure understanding of one concept before proceeding to the next. Such a controlled presentation of material is

called a "lesson".

Building lessons with traditional programming languages is tedious. Images must be constructed from the lowest level graphical primitives, sequencing has to be handled in every detail, and the interaction with the user must be written so as to cope with a great diversity of inputs. In Andrew, it is also desirable to utilize the Virtue base editor, with its typographic quality text options. Facilities to eliminate or reduce all these problems are provided directly by the CMU-Tutor programming language and development of lessons is supported by an associated programming environment.

The CMU-Tutor environment includes a lesson editor, a lesson executor, and a growing body of lessons. The executor is a portion of the lesson editor, so that while an author is preparing a lesson its pieces can be executed without waiting for a lengthy compilation. Error diagnostics indicate the exact position where an error is detected by moving the editor selection to that point. A comprehensive on-line manual is available and can be referenced by typing or pointing at a keyword.

Each piece of the lesson is called a "unit", and typically describes a single step in the sequence of images that the student will see. In some cases each unit clears the screen at its start, displays some image or animation, and asks for a student response. In other cases units are subroutines that generate some portion of an image or simply massage some data. Lessons are stored as base editor documents in a directory together with a "shell" script. To initiate a lesson, the student types a command which is actually the name of the shell script. This in turn loads the CMU-Tutor executor, giving it the lesson document and the names of any ancilliary files (fonts or data, perhaps). In the lesson document the instructions are stored in machine-independent source form and are compiled as the student executes the lesson. The compilation delay is almost invisible because of the speed of the computers used and a clever design of the executor.

We can demonstrate CMU-Tutor with the lesson *waves* developed by David Trowbridge, who has developed a number of lessons on Andrew [Trowbridge, 1986]. *Waves* teaches a student how to write equations for periodic motion such as a point on a rolling wheel. As the lesson begins, the student soon comes upon the page shown in Figure 3. Here the *Anim* button in the lower right invites the user to start an animation by pointing at the button and clicking a mouse button. Then the image of the ball in the upper right moves in a counter-clockwise circle returning to its present position at three-o'clock on the circle. (Sorry, this is not a live demonstration; no amount of clicking the *Anim* button in Figure 3 will budge the ball by even a pixel!)

After the animation, *waves* asks the user to describe the motion of the ball. The correct answer is that the ball is making a uniform, circular motion, which is the first form of equation the lesson is going to teach. When the student types a response it is echoed at the arrow just below the top left portion of the window. The response is compared by the CMU-Tutor executor against a number of possibilities anticipated by the author and the program prints a message. For this instance, the author has permitted misspellings and alternate word orders, so any answer saying something about "uniform" and "circular" will be accepted. Possible answers include "uniform circular motion", "smooth circle", "it goes around steadily" "unform circul", "round reguler".

After Figure 3, *waves* displays the "Reference Circle" shown in Figure 4. In this Figure, notice the use of boldface and alternate fonts in the text, features which are easy when the lesson is an Andrew document, but which would be tedious in a C program. Note also that the question is asking for an equation as the answer. Here again the CMU-Tutor environment simplifies the lesson author's task: the system automatically accepts such diverse replies as wt, w*t, (w)/t, wwt/w. (The equation is evaluated for a few random values of the variables and accepted if it gives the correct answer.)

The author's view of the *waves* lesson is shown in Figure 5, where the execution of the lesson is visible in the lower portion of the window. The upper portion of the window is a base editor document containing the programs source of the lesson, in this case the portion of *waves* which ~~makes the ball move in a~~ circle. When the author selects the menu option "Execute Current Unit", execution is begun for the unit whose source text in the top of the window contains the edit cursor or selection. The results of the execution are shown in the bottom of the window.

draws the reference

For image construction, the band across the middle of the window provides a choice among the most used CMU-Tutor operations. Clicking on one of these words inserts a line into the current unit with the edit cursor positioned for insertion of the arguments. In most cases the arguments are coordinate positions in the image; these are entered into the program by clicking on the image area. Each click inserts one coordinate pair in the current line, with correct punctuation. Modifying the program to change coordinates can be done by modifying the program text, but it is often easier to use a clever CMU-Tutor trick: select the coordinate pair in the lesson source and click in the lesson image. The new coordinates will replace the selected coordinates in the source and the unit will be re-executed to show the result. The author can repeatedly revise a coordinate until it is exactly right.

Once the lesson has been constructed, the author can insert a single line in the CMU-Tutor source indicating how the lesson should be resized in response to varying window size. When the lesson is executed its imagery, including font heights, is adjusted in accordance with the size of the window provided.

To give some indication of how CMU-Tutor programs can build images, Figure 6 gives an extract of *waves* showing the unit that draws the reference circle in the upper right of Figure 4.

To illustrate the CMU-Tutor facilities for answer judging, Figure 7 shows the code that evaluates the students attempt to describe the motion of the ball in Figure 3. Two commands, -arrow- and -endarrow-, are the crux of answer judging. They begin by displaying an arrow at the coordinates given and then await the student response. When the student finishes the response by typing RETURN, statements after the -arrow- up to the -endarrow- are executed. Among these are "answer judging" commands which compare the student response against some anticipated response. If a match is found, the commands indented under the judging command are executed and control skips to the endarrow. If the -answer- or -ok- commands match the response, the -endarrow- exits. If one of the other judging commands match, the -endarrow- returns control to the -arrow- so the student can try again. In the example, the operand to the -ok- command is a boolean expression which is true the third time the student makes a response; thus the student is given only three tries before the system gives the correct answer and proceeds with the lesson.

V. Conclusion

This paper has described Andrew: its file system, its user interface, and the CMU-Tutor environment. The latter provides to an author comprehensive tools for image construction and answer judging.

As do all computer users, the users of Andrew will apply it in diverse ways. Almost all will use it for communication—mail, documents—and most will use it for various computational purposes. All will use the file system to access their personal files from diverse workstations, and all will interact through the user interface.

As part of an educational institution, the members of CMU will also use computers for education: faculty will create lessons and students will take them. The CMU-Tutor environment helps both. It simplifies lesson construction and generates lessons with a consistency of user interface that enables students to concentrate on the material instead of the system commands needed to get through it.

References

[Morris, 1986] Morris, J. H., Satyanarayanan M., Conner, M. H., Howard, J. H., Rosenthal, D. S. H., and Smith, F. D. "Andrew: a distributed personal computing environment," *Comm ACM 29* (March, 1986) 184-201.

[Sherwood, 1986a] Sherwood, B. A. and Sherwood, J. N. *The CMU-Tutor Programming Language, Preliminary Edition.* Stipes Pub. Co., 10 Chester Ave., Champaign, Ill. (1986)

[Sherwood, 1986b] Sherwood, B. A. "(Workstations at Carnegie-Mellon" Proc *FJCC*, IEEE Comp. Soc. (Nov 1986) 15-17.

[Trowbridge, 1986] Trowbridge, D. "A sampler of educational software at CMU," *Proc Nat Educ Comp Conf*, San Diego (June, 1986) 135-142.

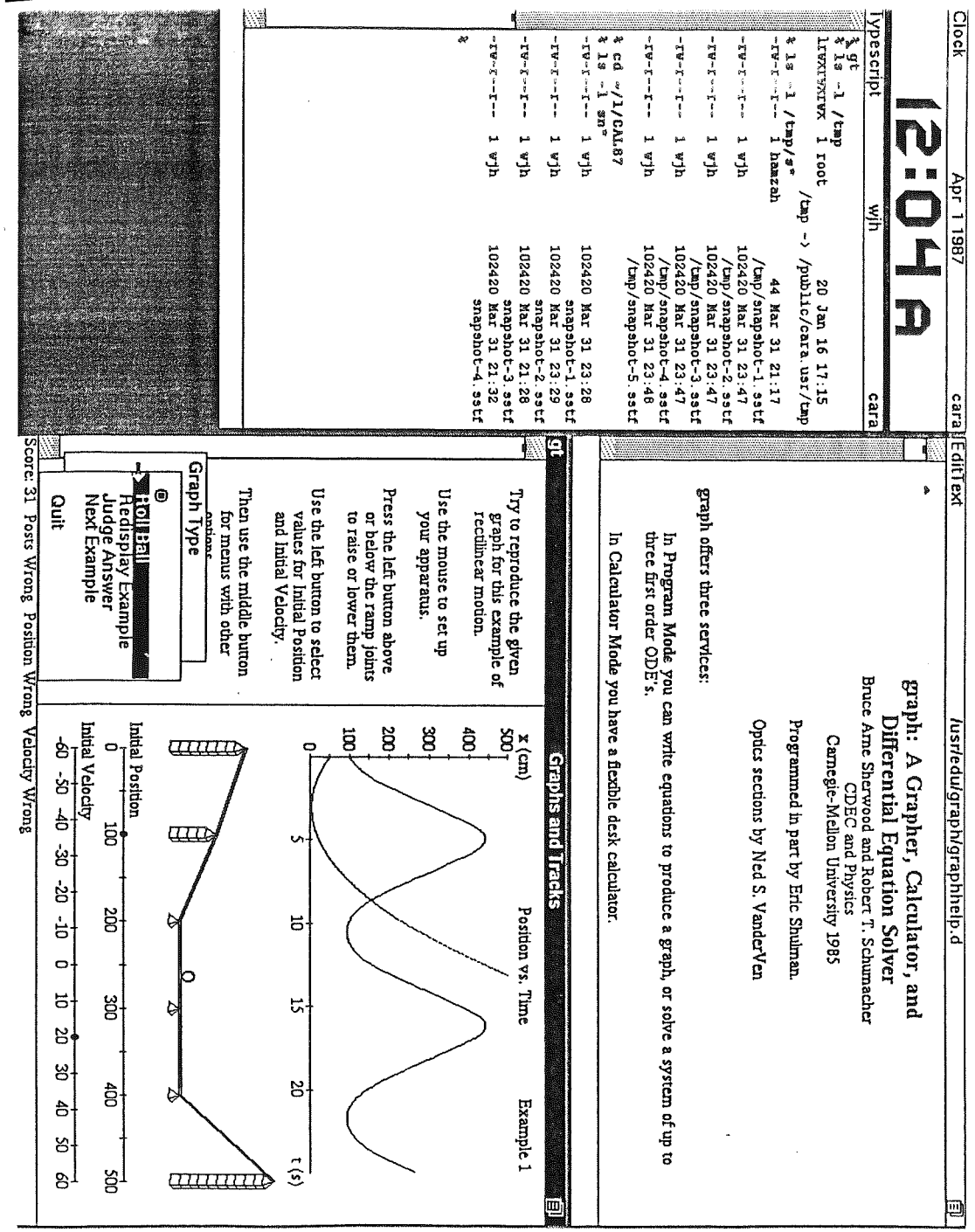

Figure 1. Andrew screen image. Clockwise from upper left the windows are: clock, text editor, *gt* (a lesson in mechanics), and typescript (a shell interface). The cursor is in the *gt* window, so its title bar is inverse video, indicating where the next key typed will go. The menu shows the "Roll ball" option and other options for *gt*.

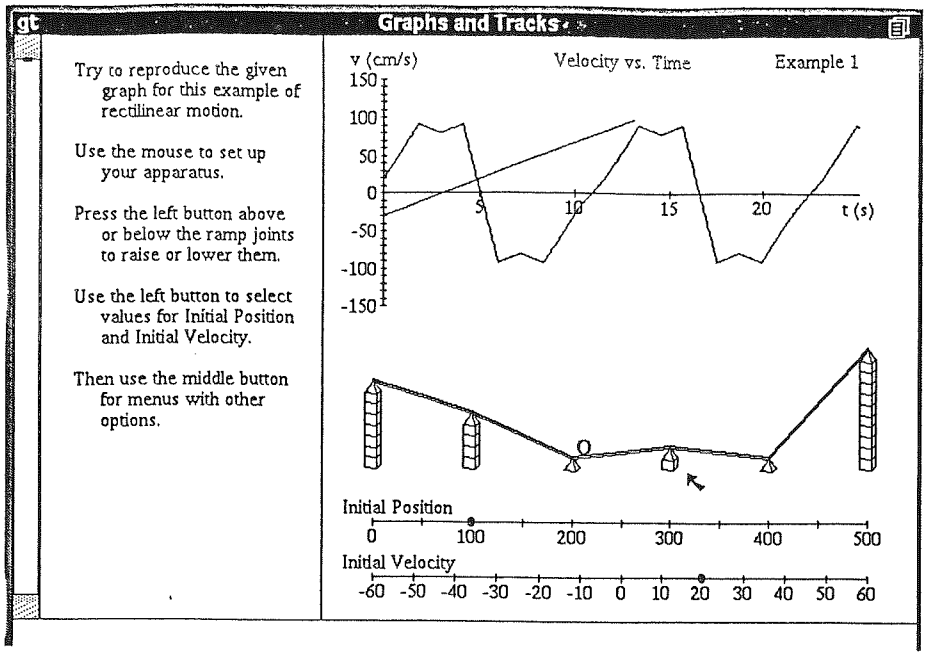

Figure 2. Gt after adjusting a post. Since the time of Figure 1, the mouse has been clicked above the fourth post, a menu option has been chosen to change the graph type to Velocity vs. Time, and Roll Ball has been selected.

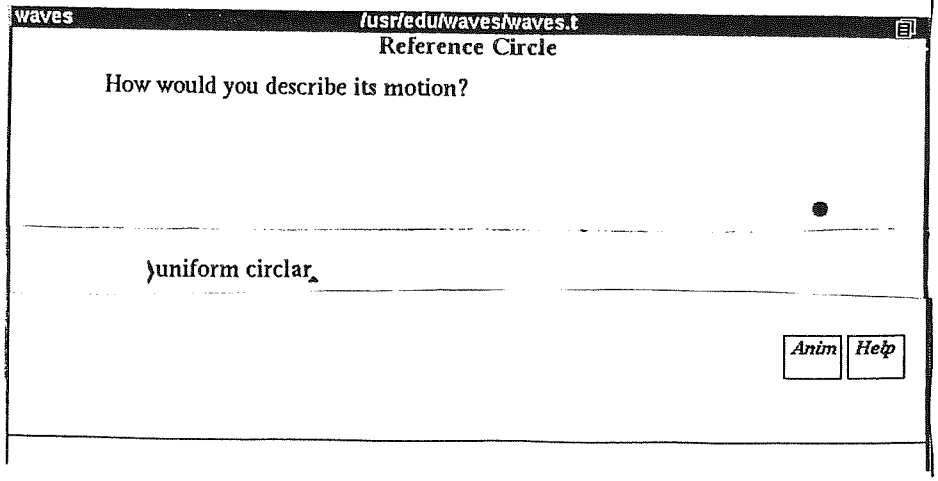

Figure 3. Snap shot of an animation. To introduce uniform circular motion, *waves* displays this image. When the student clicks on the *Anim* button the "ball" in the upper right is moved in a counter-clockwise circle 160 pixels in diameter. Then waves displays the question in the upper left. As this picture was taken, the student is answering the question after the arrow below the question. Although one correct response is "uniform circular", the response shown will be accepted because spelling correction is enabled for this answer.

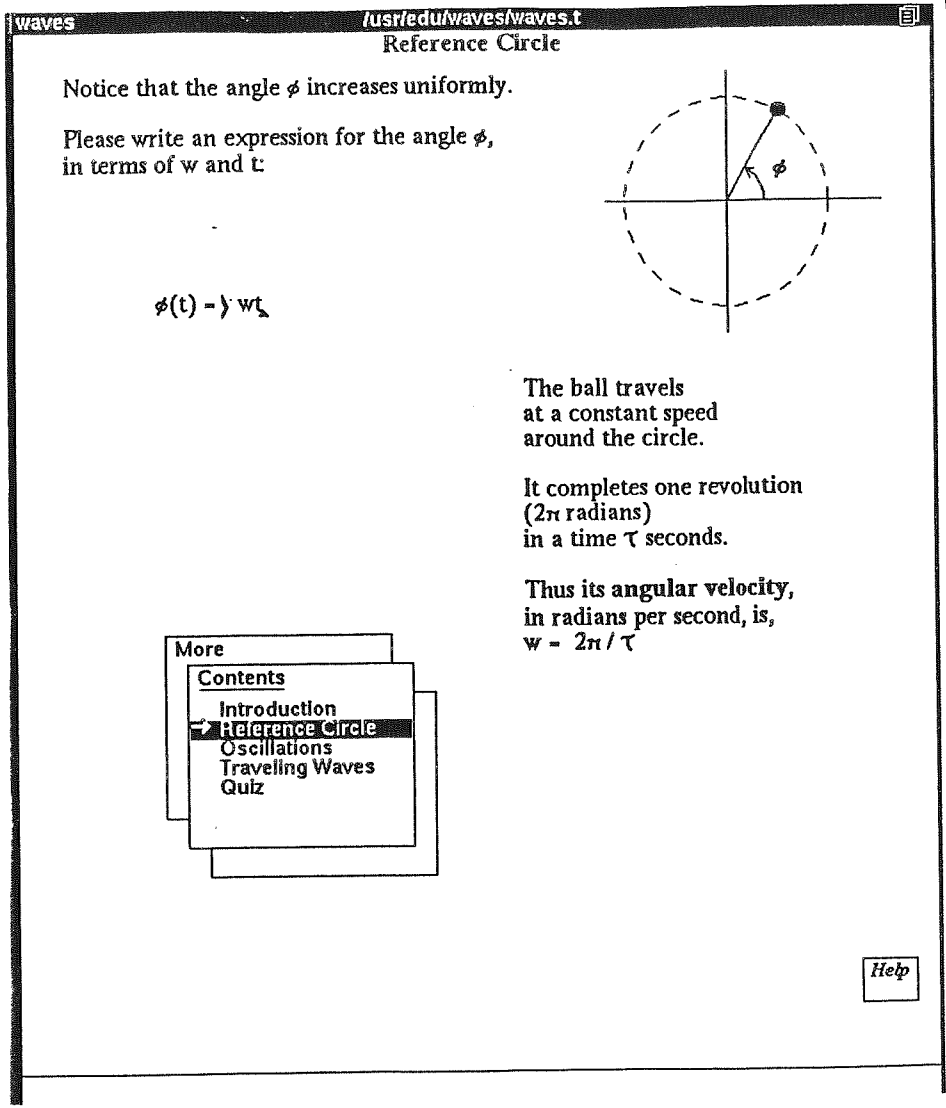

Figure 4. A complex graphic image. Note the dashed circle, curved arrow, Greek letters, and bold text. The student is answering another question at the arrow in the upper left.

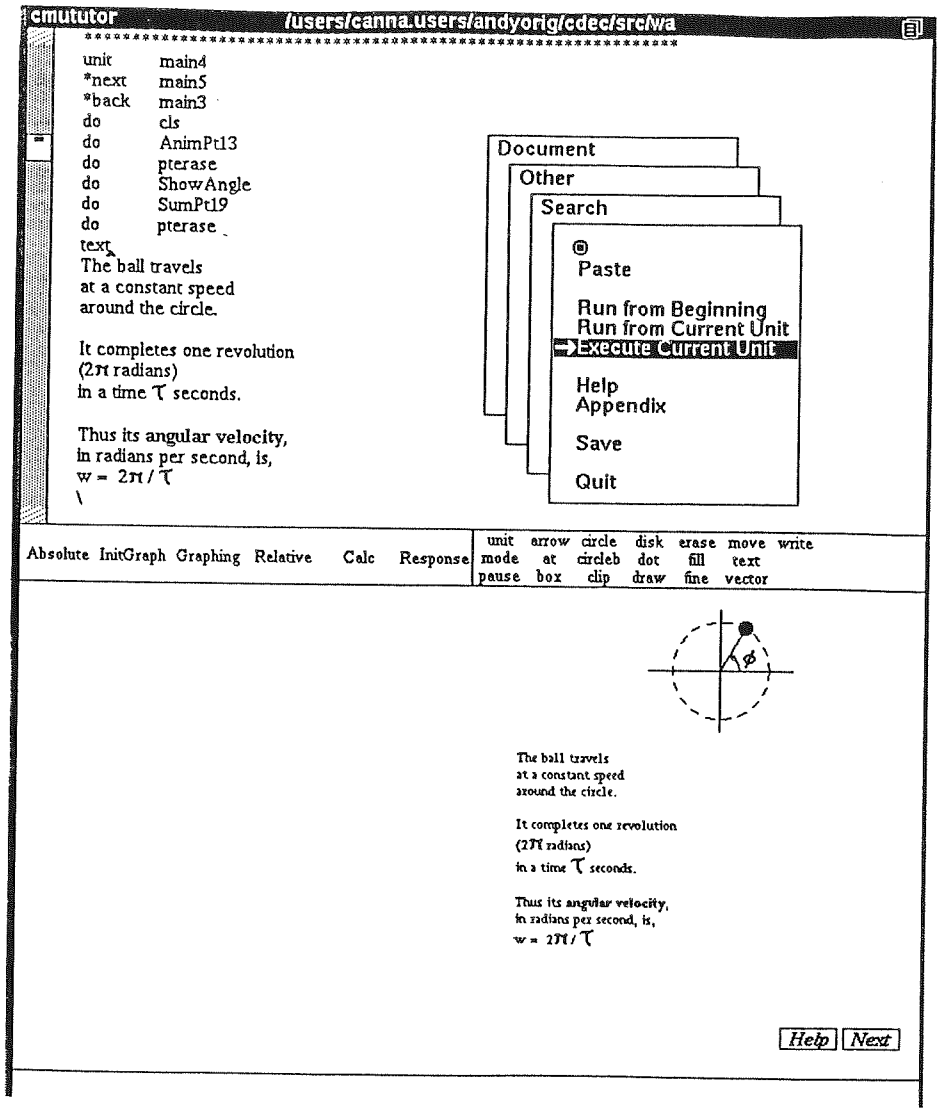

Figure 5. Creation of a CMU-Tutor Lesson. The lower portion of the window displays the execution of the lesson, the upper portion displays the source code, and the middle displays a selection of operators that create statements in the program.


```
unit      AnimPt13
calc      X1 := 650;  Y1 := 50;  X2 := 900;  Y2 := 350
          N := 13
          Yheight := (Y2-Y1)/2
          Xratio := zheight/zwidth   $$ the window's aspect ratio
gorigin   (X2+X1)/2,(Y2+Y1)/2        $$ set the origin
bounds    -(Xratio/1.2)*Yheight, -Yheight;
                    (Xratio/1.2)*Yheight, Yheight
scalex    100
scaley    100
gat       0,0                        $$ start at the new origin
*
unit      ShowAngle
calc      theta := 1.047      $$ pi/3, or 60 degrees
          rx := radius*cos(theta)
          ry := radius*sin(theta)
do        AnimPt13
gcircleb  80                         $$ draw dashed circle      <<
axes      -Xratio*Yheight, -Yheight; Xratio*Yheight, Yheight $$ <<
gdraw     0,0; rx,ry                 $$ draw radius at angle phi  <<
gat       0,0
gcircle   30,0,60                    $$ draw the curved arrow     <<
gdraw     19,17;16,25                $$    and its arrowhead    <<
gdraw     ;22,25                     $$                              <<
gat       36,35
write     φ                          $$ print the Greek phi       <<
icons     waves12
gat       rx,ry
plot      "O"                        $$ plot the ball          <<
```

Figure 6. Graphics commands. This subroutine draws the reference circle in the upper right of Figure 4. Each of the lines marked with "<<" draws a piece of the picture. Unit AnimPt613 sets the graphics origin and bounds. It is used by a number of animations so they share the same graphics portion of the window.

```
unit        GetCirc
calc        SaidUnif := FALSE
arrow       X1, Y1; X2, Y2
specs       nookno, okextra, okspell, noorder
answer      [steady steadily uniform uniformly regular smooth]
                                    [circle circular round around]
            do          GoodMsg
answer      [dont cant don't can't not no]
                                    [know say tell sure remember idea]
            do          OkayMsg     $$ "You may have noticed ..."
wrong       [smooth smoothly steady steadily uniform uniformly regular]
            do          SteadyMsg   $$ "... be more specific ..."
            calc        SaidUnif:=TRUE
wrong       [circle circular round around]
            if          SaidUnif
                        do          GoodMsg
                        judge       ok
            else
                        do          CircMsg     $$ "Yes ... and ..."
                        judge       quit
            endif
ok          zntries > 2
no
            do          DntFolMsg(zntries)      $$"We don't follow you..."
endarrow
$$
unit        GoodMsg
$$                      Print a message
text
That's correct.
\\
```

Figure 7. Unit to Accept Student Response. The correct answer is "uniform circular", but many variants are accepted. If the student says "circular" an attempt is made to get the additional information of "uniform." The various XxxxMsg units each print an appropriate message. The ok command will accept any answer if the student has formerly made two other answers. The no command is unconditional and will be executed if no other judging command matches the student's response; it prints a message that the computer does not follow the student.

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.

or

The Secretary,
Persistent Programming Research Group,
Department of Computational Science,
University of St. Andrews,
North Haugh,
St. Andrews KY16 9SS
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", University Computing, Vol. 8, N0. 2, (Summer 1986) - see PPRR-23-86.

Kulkarni, K.G. & Atkinson, M. P.
"Implementing an Extended Functional Data Model Using PS-algol", Software - Practise and Experience, Vol. 17(3), 171-185 (March 1987)

Cooper, R.L. & Atkinson, M.P.
"The Advantages of a Unified Treatment of Data", Software Tool 87: Improving Tools, Advance Computing Series, 8, 89-96, Online Publications, June 1987.

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.

Theses

The following theses, for the degree of Ph. D. unless otherwise stated, 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.

J. Livingstone
Graphical Manipulation in Programming Languages: Some Experiments, M.Sc., University of Glasgow, 1987

Persistent Programming Research Reports

This series was started in May 1983. The following list gives those which have been produced at 9th July 1986. Copies of documents in this list may be obtained by writing to the addresses 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-12-87	PS-algol Reference Manual - fourth 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-24-86	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-27-87	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. & Morrison, R.	£1.00
PPRR-32-87	Polymorphism, Persistence and Software Reuse in a Strongly Typed Object Oriented Environment - Morrison, R, Brown, A, Connor, R and Dearle, A	£1.00
PPRR-33-87	Safe Browsing in a Strongly Typed Persistent Environment - Dearle, A and Brown, A.L.	£1.00

PPRR-34-87	Constructing Database Systems in a Persistent Environment - Cooper, R.L., Atkinson, M.P., Dearle, A. and Abderrahmane, D.	£1.00
PPRR-35-87	A Persistent Architecture Intermediate Language - Dearle, A.	£1.00
PPRR-36-87	Persistent Information Architectures - Atkinson, M.P., Morrison R. & Pratten, G.D.	£1.00
PPRR-37-87	PS-algol Machine Monitoring - Loboz, Z.	£1.00
PPRR-38-87	Flexible Incremental Bindings in a Persistent Object Store - Morrison, R., Atkinson, M.P. and Dearle, A.	£1.00
PPRR-39-87	Polymorphic Persistent Processes - Morrison, R., Barter, C.J., Brown, A.L., Carrick, R., Connor, R., Dearle, A., Hurst, A.J.and Livesey, M.J.	£1.00
PPRR-40-87	Andrew, Unix and Educational Computing - Hansen, W. J.	£1.00
PPRR-41-87	Factors that Affect Reading and Writing with Personal Computers and Workstations - Hansen, W. J. and Haas, C.	£1.00
PPRR-42-87	A Practical Algebra for Substring Expressions - Hansen, W. J.	£1.00
PPRR-43-87	The NESS Reference Manual - Hansen, W. J.	£1.00