A HYPERTEXT ENVIRONMENT FOR UNIX

Przemyslaw Prusinkiewicz and *James Hanan*

Department of Computer Science
University of Regina
Regina, Saskatchewan, Canada S4S 0A2

Build on the work of others.
Henry McGilton, Rachel Morgan:
Introducing the UNIX system

ABSTRACT

This paper describes the design of a hypertext environment for UNIX. The main thesis is that a powerful hypertext system can be simply created by adding a few programs to a typical UNIX environment with a window management system.

RESUME

Dans cet article nous présentons un environnement hypertexte conçu pour UNIX. La thèse principale est qu'un tel environnement peut être construit en ajoutant quelques programmes très simples au système d'exploitation UNIX doté d'un système de gestion de fenêtres.

1. INTRODUCTION

What is a hypertext system? An old definition [Nelson 1967, see also Conklin 1987] describes it as "a combination of natural language text with the computer's capacity for interactive branching, or dynamic display, of a nonlinear text which cannot be conveniently printed on a conventional page." Newer systems complement this definition by emphasizing the multimedia aspect of hypertext; the combination of text with images, animated scenes, and audio synthesis [see, for example, Feiner, Nagy and Van Dam 1982, Yankelovich, Meyrowitz and Van Dam 1985, Halasz, Moran and Trigg 1987, Atkinson et al. 1987, Goodman 1987, Yankelovich et al. 1988]. The images can be prestored raster files, or they can be created in real time by programs which are automatically called when the corresponding text segment is selected for viewing. In this case, the system runs various programs implicitly selected by the user who browses through the database. This assistance in selecting programs is a function of hypertext known as task switching [Card and Henderson 1987].

The hypothesis explored in this paper is that a powerful hypertext system can be created by adding a few programs to a typical UNIX environment with a window management system. We will illustrate this concept with the example of a hypertext system called **metatext**, which was implemented on a Silicon Graphics IRIS 3130 workstation running UNIX System V and the **mex** window management system [Silicon Graphics 1987a]. The same approach could be applied to other versions of UNIX and other window management systems.

The idea of building a hypertext system on top of a standard UNIX environment draws on some concepts already present in the literature. Specifically, Witten and Bramwell [1985] developed a UNIX-based system for viewing documents in a nonlinear way. However, they did not exploit the possibility of automatically calling arbitrary UNIX programs from the documents. On the other hand, the idea of considering an interactive document as a script which may refer to any program available on the system was implemented in the UNIX educational program **learn** [Kernighan and Lesk 1979]. To some extent **metatext** is a descendant of these two approaches.

The paper is organized as follows. After a general description of the **metatext** structure (Section 2), we specify two essential components of the system: the *frame* (Section 3) and the **browse** program (Section 4). Next, (Section 5) we present three example applications of **metatext**. We conclude (Section 6) by summarizing our experience with **metatext** and outlining some problems for future research.

2. THE STRUCTURE OF METATEXT

The structure of **metatext** can be described from two perspectives: the static structure of links between information nodes and the dynamic structure of processes created while browsing and displaying information.

A **metatext** data base (Fig. 1) consists of two types of files: frames and index files. A *frame* is the basic information and display unit. It contains text to be displayed in the text window, and a list of commands to be run along with the text. An *index file* is a list of names grouping related frames into a *section*. A given frame can be listed in several index files, and can include references to other index files. This establishes a graph of references not limited to hierarchical structures, built on top of the UNIX file system.

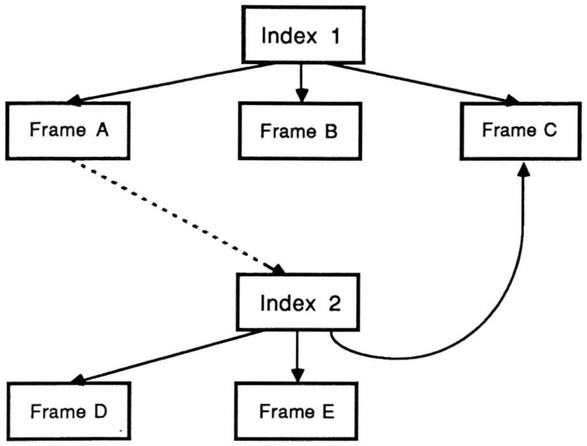


Figure 1.
Static structure of a **metatext** data base.

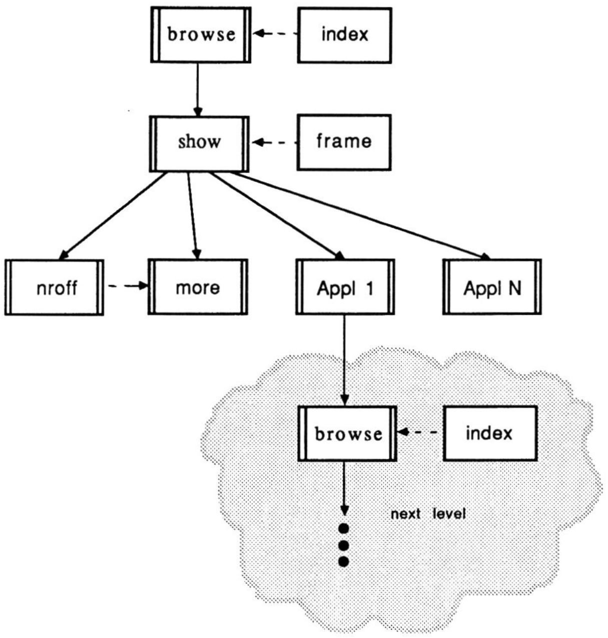


Figure 2.
Dynamic structure of **metatext** processes.

Metatext runs as a number of concurrent processes (Fig. 2). They can be divided into three classes:

- **Browse** processes which read index files and allow the user to interactively select a frame.
- **Show** processes which are spawned by a **browse** process once a selection has been made. Each **show** process reads the frame passed as its argument, and creates child processes which display the frame text and execute the accompanying commands.
- Application processes corresponding to the commands listed in the frame.

It is essential to the concept of **metatext** that new **browse** processes can be created in the same way as any application process, by executing an appropriate command included in the frame. Consequently, several **browse** processes may be active at the same time. For example, consider a hierarchical organization of a **metatext** data base. When navigating from the root towards the leaves of the data base, new **browse** processes are created to handle frame selection at successively lower levels. On the other hand, when the user navigates back towards the root, the lower level processes are no longer necessary and can be killed. Thus, the number of active **browse** processes changes dynamically, reflecting the user's current position within the data base. It follows that the navigation mechanism in **metatext** is inherently based on the concept of multiprogramming.

3. IMPLEMENTING A FRAME USING THE SHELL

Consider the operation of displaying a section of text with accompanying illustrations. The text to be displayed as well as the names of all data and program files involved are included in a single frame. An example frame, called *L_system_overview*, is given below.

```
.pl 30
.ce
\fIGraphical Applications of L-systems \fR
.LP
A string of symbols generated by an L-system can be
used to create a picture if the string symbols are
interpreted as commands controlling a Logo-style
turtle. L-systems can generate a large variety of
images: fractals, such as the Menger \fIsponge\fR;
kolam patterns (folk art from southern India),
such as \fIscissors\fR; developmental plant models,
such as \fIlychnis\fR; and realistic plant images,
such as the \fItulip\fR. The program developed
to perform this task is called "\fIp\fRlant and
\fIf\fRractal \fIg\fRenerator" or \fIpfg\fR.

:cd ~/PATTERNS
:ipaste menger.ras  # load Menger sponge image
:pfg scissors.l scissors.v  # generate scissors kolam
:cd ~/PLANTS
:pfg lychnis.l lychnis.v lychnis.a  # animate model
:pfg tulip.l tulip.v  # generate a tulip image
```

The text is specified in **nroff** format. Programs which should be automatically invoked to illustrate the text are listed, together with their parameters, in the lines starting with a colon.

In order to display such a frame, the following actions must be performed:

- Read the specified frame.
- Filter the text, then format according to the target window size and display using a scrolling program (the text need not fit in a single window).
- Filter the command lines from the textual content. Run all required programs, with their corresponding parameters, concurrently.

The shell script **show**, which performs these tasks, is given below:

```
clear > /dev/console
grep -v \^: $1| nroff -ms | more -l > /dev/console
sed -n '/^:/s///p' $1 | sh
```

The first line clears the text window (in this case the console). The next line filters the text using **grep** and pipes it to **nroff** for formatting, then to **more** for display in the text window. Finally, the command lines are isolated, their leading colons are stripped using **sed** and they are passed to **sh** for processing.

The result of running the command

```
% show L_system_overview
%
```

is shown in Fig. 3. All information specified in the frame has been presented. Note that the graphic images can be further manipulated (as indicated by the presence of a menu) since the processes associated with each window remain active until explicitly killed.

This example shows that a very simple shell script makes it possible to associate text with arbitrary programs. For example, these programs may generate images illustrating the text. If a number of frames following the same format are available, each of them can be displayed using the **show** command typed from the console. Obviously, this is not a convenient method for browsing through a collection of frames. The next section presents a browsing utility which provides a better solution to the navigation problem.

4. NAVIGATING THROUGH A COLLECTION OF FRAMES

A utility called **browse** was developed to assemble a set of frames into a linked system. **Browse** is invoked with a text file as its argument. Each line of this file is entered as a separate item in a pop-up menu. Usually, the input file is an index of the frames composing a section.

A running **browse** process is represented on the screen as a small (approximately 2 x 5 cm) window, with a name corresponding to the index name (Fig. 4). If the user selects this window, the menu containing the index appears. By selecting an item, the user causes **browse** to fork a **show** process with the indicated frame passed as the argument.

In addition to the frame names read from the index file, the **browse** menu contains three default items. The items *next* and *previous* are used to step through the list of frames in forward or reverse order, respectively. The *quit* item kills the **browse** process.

In spite of its simplicity (250 lines of C code, including comments), the **browse** program is a powerful tool for interconnecting frames and index files into a network. While a single **browse** process performs the elementary operation of mapping an index file into a set of frames, a set of concurrent **browse** processes implements a flexible, non-linear access method to the **metatext** frames.

5. APPLICATION EXAMPLES

In this section we present three applications of **metatext** to illustrate the potential of the system.

5.1. Organization of computer experiments

The original purpose of **metatext** was to organize descriptions, programs, data and results of computer simulations of plant development. A problem was created by the large number of files (several hundreds) involved in the experiments. **Metatext** groups these files into frames. A typical frame provides a description of the experiment and embeds the program calls with appropriate parameters which are involved in the corresponding simulation. Related experiments are listed in a common index file. Lower-level index files are accessed primarily from a top-level index, although cross-references between frames from several index files are also occasionally used. The task-switching capabilities of **metatext** are used extensively to organize experiments involving several programs which cannot run concurrently (for example, the growth-simulation program and the patch editor used to define plant organs).

The user can modify simulation parameters, animate development processes, edit their descriptions, create new experiments, include them in the system, and delete obsolete frames. Thus, the **metatext** data base constantly evolves, reflecting the current state of understanding of the research problem considered and the corresponding evolution of the underlying software. For further examples of hypertext application to software development see [Bigelow 1988].

5.2. A computer-assisted geometry course

Another application being explored is the use of **metatext** to organize a computer-assisted geometry course. The main application used for this purpose is an interactive graphics program called **L.E.G.O.** [Fuller, Prusinkiewicz and Rambally 1985]. **L.E.G.O.** provides an electronic metaphor for classic Euclidean constructions with compass and ruler. They can be interactively defined, saved, and used as elements of more complicated constructions. Thus, **L.E.G.O.** creates a "mathematical microworld" [Kearsley 1987] in which experiments take place. In this context, the purpose of **metatext** is to guide the student from one construction to another following the logical structure of Euclidean geometry. The student starts from the simplest constructions, such as the bisection of a line and a wedge, and uses them in a hierarchical way to solve more and more complex problems. Consequently, the student learns geometry by recreating these constructions in the appropriate order under the guidance of **metatext**.

5.3. Organization of IRIS demonstration and tutorial programs

Bundled with the IRIS workstation are a number of demonstration and tutorial programs. We have used **metatext** to organize these programs into a structure which allows the beginner to explore the IRIS environment easily. For instance, the on-line tutorial programs referenced in the *IRIS Programming Tutorial* [Silicon Graphics 1987b] are grouped together in the *tutorial* index file for easy access while reading the manual. Some of these programs are also cross-listed in other index files. For example, the program **queue** is cross-listed in the index which groups all programs demonstrating interface handling on the IRIS. This cross-listing allows the user to browse through the network of programs, exploring interests as they occur.

6. CONCLUSIONS

UNIX with a window management system supports most hypertext functions. A hypertext environment can be composed from standard UNIX utilities with the addition of a browsing program. This design approach is consistent with the UNIX philosophy of supporting new functions by combining existing utilities. From that point of view, a parallel can be drawn between **metatext** and the first version of the **spell** program. Like **spell**, reportedly written in one afternoon [Bentley 1985], a useful hypertext environment for UNIX can be developed in a matter of days.

Naturally, embellishments and enhancements would require additional work. Two problems open for further research are presented below.

- *Design of a window management system with support for hypertext.* Consider the problem of calling an interactive program from a frame. When run standalone, this program may expect the user to specify a window position and select the desirable menu options using a mouse. On the other hand, when using hypertext it may be preferable to have the corresponding parameters predefined within the frame. This can be accomplished in an elegant and general way if the hypertext system has a mechanism for simulating user operations performed with the mouse. Technically, this requires the **show** process to insert data on the mouse event queue of an application process. Unfortunately, in the current implementation of **mex** such inter-process queue insertions are not possible.
- *Development of hypertext-oriented utilities and programs.* The user of a hypertext system may be required to modify the frame content. Naturally, such changes can be achieved using an editor. The problem is that the standard UNIX editors do not contain any provisions which would make modifications to some text fields feasible while protecting other fields from changes. Thus, if the user is allowed to change some data in the text, he or she can also alter the entire text. In some applications, such as computer-assisted instruction, this is clearly undesirable (the student may be required to insert notes but not alter the lesson content). Consequently, a text editor should be developed to allow modification of selected fields in a file, while protecting others.

The above examples illustrate a more general problem. A window manager or the UNIX utilities may lack some features which would be useful in hypertext applications. It would be worthwhile to identify all such features in order to provide guidelines for future extensions of the system software aiming at better support for hypertext.

We have found **metatext** to be a very useful tool, particularly in its ability to help organize computer experiments. It has provided us with a flexible hypertext environment at a very reasonable cost.

ACKNOWLEDGMENTS

Peter Broadwell clarified the possibilities and limitations of **mex**. Reviewers provided very helpful comments which have been incorporated into the paper. This work could not have been done without the support and facilities provided by the Department of Computer Science at the University of Regina. Partial support from the Natural Sciences and Engineering Research Council of Canada (grant number A0324) is also gratefully acknowledged.

REFERENCES

Atkinson, B., *et al.* [1987]: *HyperCard.* Software for Macintosh microcomputers.

Card, S. K., and Henderson, A. [1987]: A multiple, virtual-workspace interface to support user task switching. *CHI + GI '87 Conference Proceedings*, pp. 53-59.

Bentley, J. [1985]: Programming pearls: A spelling checker. *Communications of the ACM* **28**, No. 5, pp. 456-462.

Bigelow, J. [1988]: Hypertext and CASE. *IEEE Software*, March 1988, pp. 23-27.

Conklin, J. [1987]: Hypertext: An introduction and survey. *Computer* **20**, No. 9, pp. 17-40.

Feiner, S., Nagy, S., and van Dam, A. [1982]: An experimental system for creating and presenting interactive graphical documents. *ACM Transactions on Graphics* **1**, No. 1, pp. 59-77.

Fuller, N., Prusinkiewicz, P. and Rambally, G. [1985]: L.E.G.O. – An interactive system for teaching geometry. *World Conference on Computers in Education*, Norfolk, Virginia, pp. 359-364.

Goodman, D. [1987]: *The complete HyperCard handbook.* Bantam Books, New York.

Halasz, F. G., Moran, T. P., and Trigg, R. H. [1987]: NoteCards in a nutshell. *CHI + GI '87 Conference Proceedings*, pp. 45-52.

Kearsley, G. [1987]: *Artificial intelligence and instruction – applications and methods.* Addison-Wesley, Reading, Massachusetts.

Kernighan, B. W., and Lesk, M. E. [1979]: LEARN – computer-aided instruction on UNIX. In *UNIX user's supplementary documents*, 4.3 bsd, University of California, Berkeley 1986.

Nelson, T. H. [1967]: Getting it out of our system. In: G. Schechter (Ed.): *Information retrieval: A critical review.* Thompson Books, Washington, D.C.

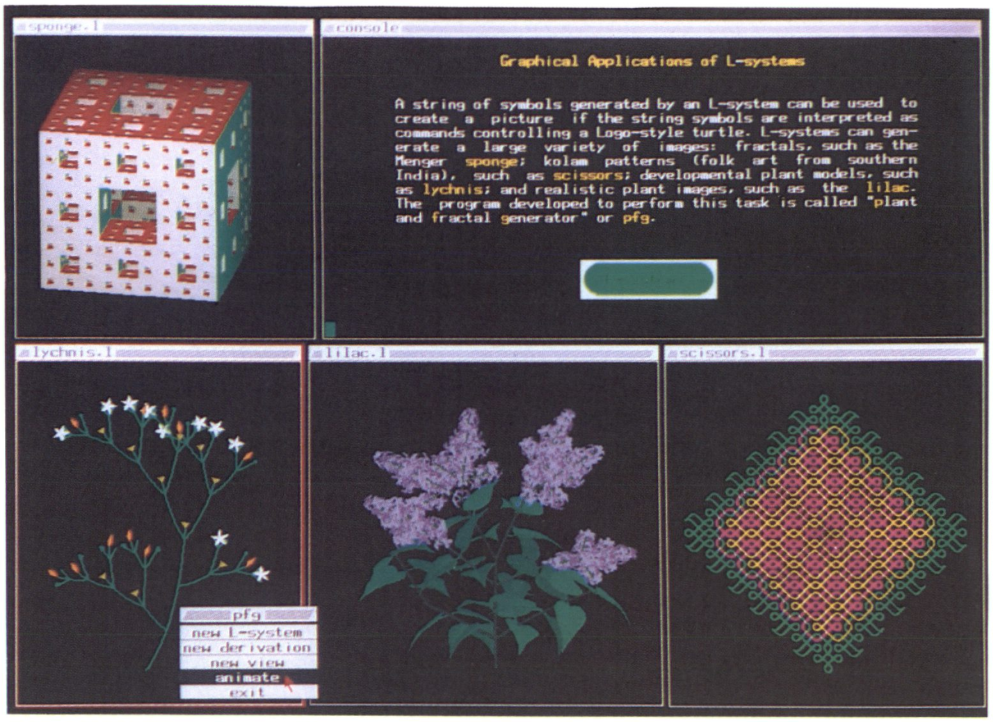
Silicon Graphics [1987a]: *IRIS user's guide volume I: Graphics programming.* Version 3.0. Mountain View, California.

Silicon Graphics [1987b]: *IRIS programming tutorial.* Mountain View, California.

Witten, I. H., and Bramwell, B. [1985]: A system for interactive viewing of structured documents. *Communications of the ACM* **28**, No. 3, pp. 280-288.

Yankelovich, N., Haan, B. J., Negrowitz, N. K., and Drucker, S. M. [1988]: Intermedia: The concept and construction of a seamless information environment. *Computer* **21**, No. 1, pp. 81-96.

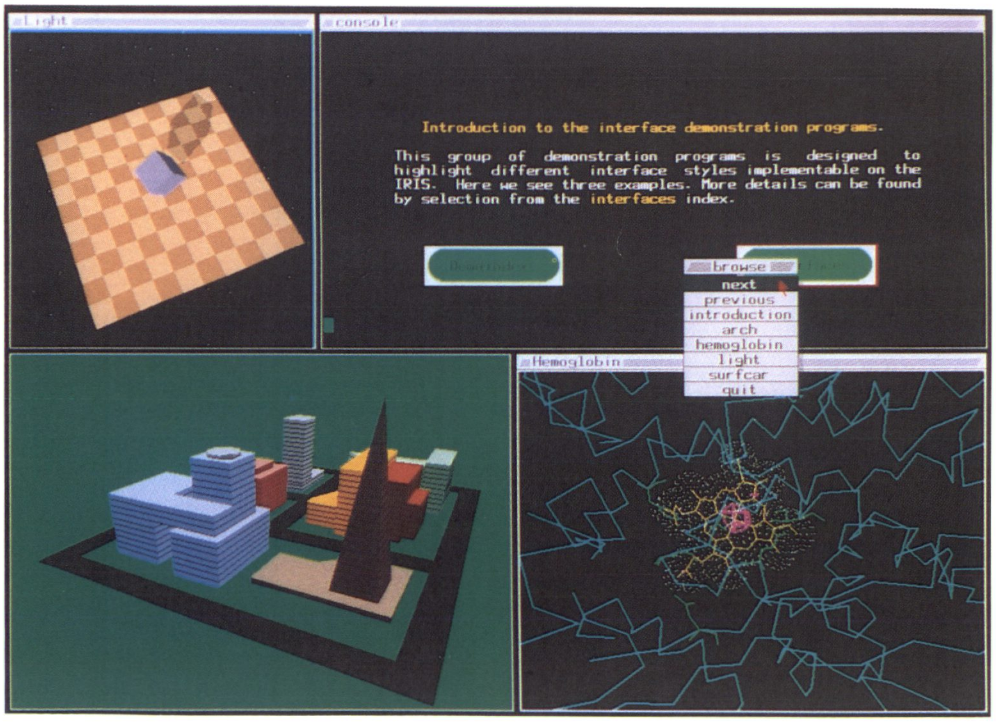
Yankelovich, N., Meyrowitz, N., and van Dam, A. [1985]: Reading and writing the electronic book. *Computer* **18**, No. 10, pp. 15-30.



Graphical Applications of L-systems

A string of symbols generated by an L-system can be used to create a picture if the string symbols are interpreted as commands controlling a Logo-style turtle. L-systems can generate a large variety of images: fractals, such as the Menger sponge; kolam patterns (folk art from southern India), such as scissors; developmental plant models, such as lychnis; and realistic plant images, such as the lilac. The program developed to perform this task is called "plant and fractal generator" or pfg.

Figure 3. Example of a **metatext** frame display. This frame belongs to the **plant development** database discussed in Section 5.1.



Introduction to the interface demonstration programs.

This group of demonstration programs is designed to highlight different interface styles implementable on the IRIS. Here we see three examples. More details can be found by selection from the interfaces index.

Figure 4. Example of a **metatext** frame display. The small windows at the bottom of the text window represent two active **browse** processes. The frame belongs to the *IRIS demonstrations* data base outlined in Section 5.3.