A HYPERTEXT ENVIRONMENT FOR UNIX

Przemyslaw Prusinkiewicz and James Hanan

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

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.

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.
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 \fIsissors\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 \fIflip\fRlant and \fIfractal \fIg\fRenerator" or \fIfr\fR.

.cd /PATTERNS
:paste 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
```

Graphics Interface ’88
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 \^$| 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.

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 database 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 composed from standard UNIX utilities with the addition of a browsing program 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.

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.

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


Graphics Interface '88


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


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

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.

Graphics Interface '88