HTEXT: AN INTUITIVE, FILE-TRANSPARENT HYPERTEXT SYSTEM

by

Narciso B. Jaramillo, Michael Schiff, and Lawrence A. Rowe

Memorandum No. UCB/ERL M91/21

26 February 1991

HTEXT: AN INTUITIVE, FILE-TRANSPARENT HYPERTEXT SYSTEM

by

Narciso B. Jaramillo, Michael Schiff, and Lawrence A. Rowe

Memorandum No. UCB/ERL M91/21

26 February 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Htext: An Intuitive, File-Transparent Hypertext System

Narciso B. Jaramillo, Michael Schiff, and Lawrence A. Rowe*
Computer Science Division — EECS
University of California
Berkeley, CA 94720

February 26, 1991

Abstract

Htext is a simple, intuitive hypertext system for Unix that runs under X Windows and is written in C++ using the InterViews toolkit. This paper describes the system's user interface, which incorporates several novel link display, browsing, and window management features. It also discusses how Htext achieves *file-transparency*, the ability to link together pre-existing text files without modifying the files themselves.

Keywords: hypertext, user interfaces, X Windows

1 Introduction

This paper describes the design and implementation of a simple, intuitive hypertext system, Htext, for Unix[1] and X Windows. The system is designed to take advantage of existing on-line text. It is intended primarily as a tool for linking documents together, rather than as a system for facilitating collaborative work. We call it *file-transparent* because it makes no modifications to existing text files in order to link them into hyperdocuments. File-transparency allows users to

- Work with files containing source code, text formatter input, and so forth without disturbing their contents.

*This material is based upon work supported under National Science Foundation Graduate Fellowships awarded to the first two authors.

[1]Unix is a trademark of Bell Laboratories.

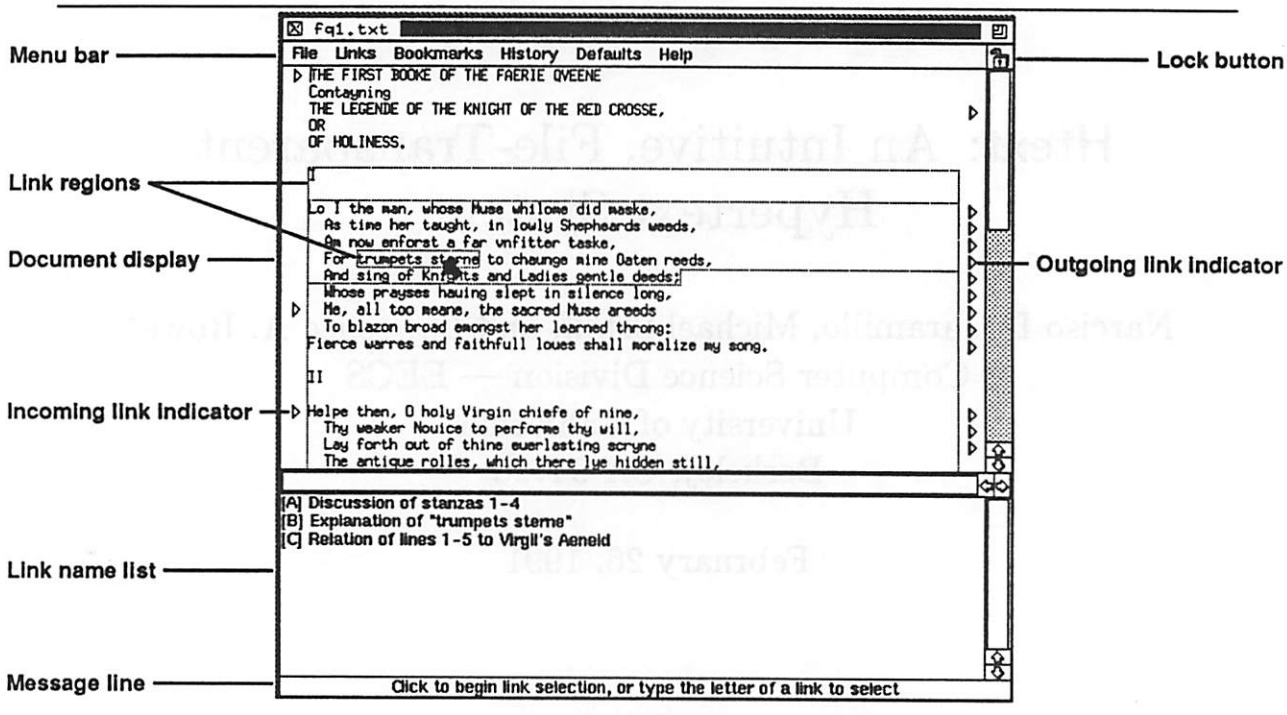


Figure 1: An Htext window

- Work with files which they do not own but can still read (for instance, files in public text databases). We wanted the users to be able to link these files to other documents.
- Use existing document files at their normal locations within the Unix directory hierarchy. We wanted to allow users to open existing documents with a normal filebrowser dialog box.

Htext fits into the mainstream of existing hypertext systems [1]. However, our goals in designing Htext diverged from those of systems such as NoteCards [3], NLS/Augment [2], and Intermedia [5]. In particular, one of our highest priorities in designing Htext was the ability to work with existing files without modifying them to include link information. We were also interested in providing the capability to work easily with long, internally-structured documents as well as with short ones. Since we assumed that such long documents would not necessarily have been created to work with a hypertext system, and thus would not always be easily separable into hypertextual chunks, we wanted to provide very general, possibly overlapping, links between arbitrary regions of the documents.

A typical Htext window is shown in Figure 1. In the upper part of the window, the *document display* contains the document associated with the window. Within the document display, several *link indicators* (displayed as triangles) show where links enter and leave the text. The document display also contains grey outlines, which

indicate important regions of text that the mouse is currently over; links pertinent to those regions are displayed in the lower half of the window, forming the *link description list.* Below the link description list is a message line containing information about the user's current action, or brief directions about how to proceed.

Section 2 of this paper describes the hypertext model we adopted in designing Htext. Section 3 describes the document browsing and editing interface. Section 4 contains some notes on Htext's implementation. Section 5 discusses Htext's current status and possible future extensions.

2 Htext's model

The hypertext model underlying Htext is straightforward and intuitive. From the user's point of view, there are only three entities: documents, regions within documents, and links between documents. A *region* is a contiguous string of characters that may span several lines of text; in the document display, active regions (i.e., regions that contain the mouse cursor) are outlined by light grey lines (see Figure 1). Each link has a source region in a document and a destination region in the same or in a different document. Each link is directed in that its descriptions at its source and at its destination are different from each other. However, users can follow links forward (from the source to the destination) as well as backward (from the destination to the source). The directionality thus provides orientation cues for users without limiting the ways that they can browse through connected documents.

In systems such as NoteCards and NLS/Augment, links connect a single point in the text of one document to the entire text of another document. In contrast, Htext uses regions as the sources and destinations of links, and allows these regions to overlap. Regions are more informative than single points in that they show users precisely what pieces of text are linked. Links in Htext can exist between words, sentences, paragraphs, or even larger pieces of text. For example, a sentence in one paper might refer to two paragraphs in another paper; the user could link the sentence to the paragraphs. Furthermore, for heavily annotated texts with many link sources, link regions help to distinguish one link from another. Despite the complications involved in displaying overlapping link regions, we feel that this feature is essential.

Systems such as Textnet, IBIS, and SYNVIEW [1] that are intended for use in a collaborative environment benefit from typed links (e.g. AGREE, DISAGREE, SUPPORT, REFUTE) that support particular structurings of a hyperdocument. However, a limited set of types also limits the generality of the system. Htext is designed primarily for linking together heterogeneous, existing documents to form various structures, not for assisting in collaborative work of any particular type. Thus, it allows users to name links arbitrarily without forcing them to choose from a restricted set of link types. Named links can communicate a variety of relationships between linked texts, providing important information for users who are browsing documents.

3 The browsing and editing interface

This section describes the interface Htext provides for browsing and editing documents and links. Section 3.1 describes the way the system displays link regions and allows the user to follow links between them. Section 3.2 describes Htext's facilities for managing document windows and avoiding screen clutter. Section 3.3 describes two orientational mechanisms, bookmarks and history, that the system provides. Section 3.4 describes how the user creates new links and changes existing ones. Finally, section 3.5 describes the text editor that is built into Htext.

3.1 Link display and browsing

As mentioned earlier, we consider overlapping link regions to be essential for Htext. Unfortunately, it is difficult to display multiple link regions in a single area simultaneously and cleanly. We also believe that almost any significant markings within the text itself are disruptive, especially in long, coherent documents. Our solution to this problem is to display *link indicators* outside the text, and to display link region boxes within the text only when the user explicitly requests them.

Link indicators appear as rightward-pointing triangles in the margins of the text (see Figure 1). Links *to* a document (i.e. those with destination regions in the document) are indicated by rightward-pointing triangles in the left margin; links *from* the document are indicated by rightward-pointing triangles in the right margin. Each link indicator is aligned with the vertical center of its associated link source and destination regions. These indicators give users a rough idea of where the links are in document without distracting them while they are reading the text.

When the user moves the mouse over a link region or regions in the document, or over a link indicator attached to a set of link regions, those link regions are outlined by gray boxes. No special provision is made for overlapping regions—all of the boxes for all of the regions containing the point in the text specified by the mouse cursor are displayed. Simultaneously, the names of all of the relevant links are displayed in the *link name list* below the text.

Browsing in Htext is tightly integrated with this method of link display (see Figure 2). While a particular set of link regions is boxed in the text and their associated links are shown in the link name list, the user may follow a link by one of several methods.

Initially, all of the link names are displayed unhighlighted with letters next to them in brackets. Typing the letter associated with a link highlights the name of that link in the link name list. Alternatively, clicking the mouse highlights the first link in the link name list. After either of these actions, the link name list is "locked": link names will remain displayed even if the cursor is moved out of the link region. Different links may be selected by clicking on a link name with the mouse or by pressing the letter

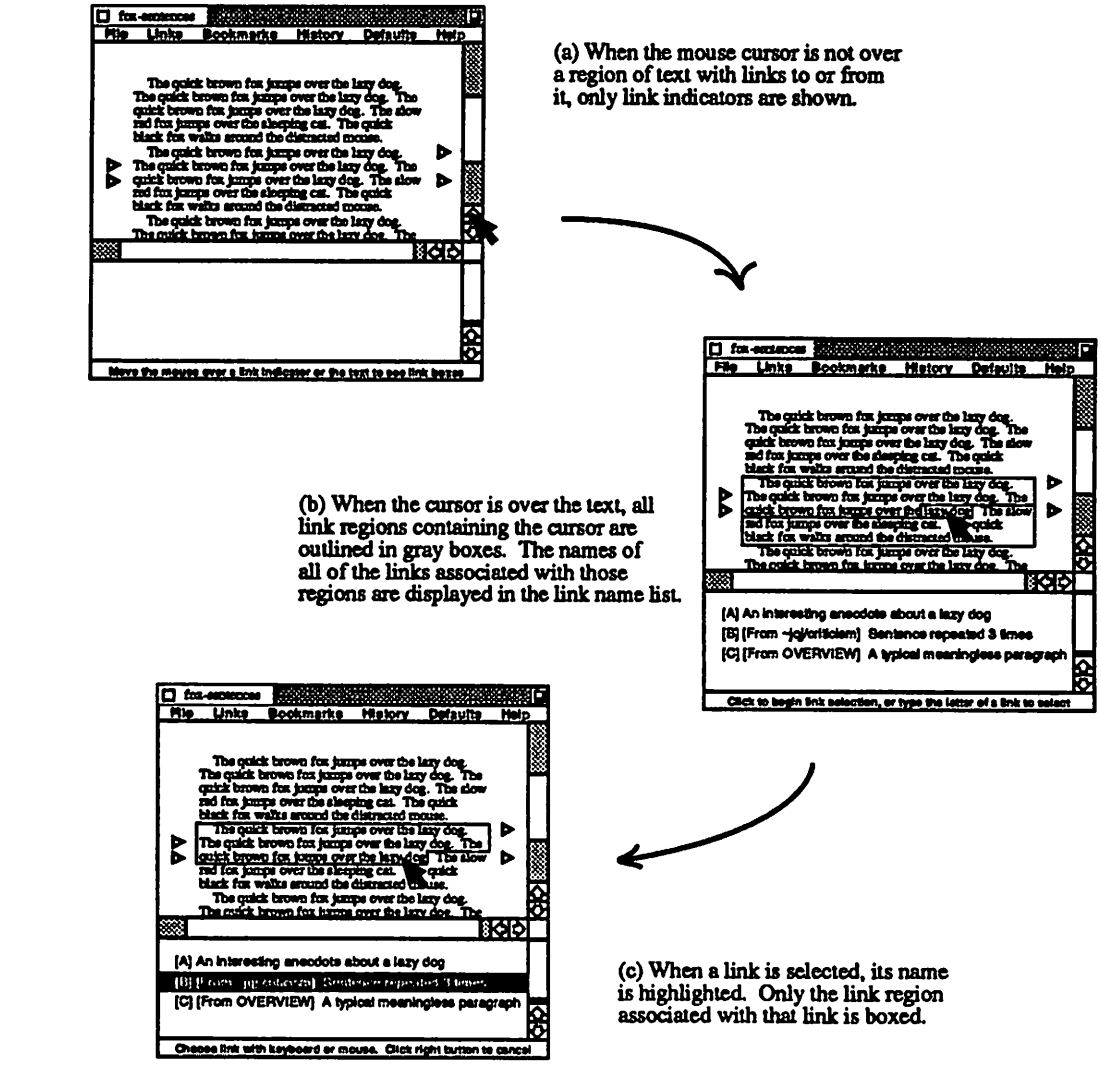


Figure 2: Browsing in Htext

associated with a link on the keyboard. Only one link may be highlighted at any given time; while a link name is highlighted, only the box around that link's associated link region is displayed. This helps the user to select the most interesting or appropriate link. After the chosen link has been highlighted, clicking the mouse anywhere in the text will follow it. The system creates a window for the destination document if one does not already exist, raises the window, and blinks a box around the destination link region.

We have found this browsing method to be simple and flexible. For a region with only one link, double-clicking over the link region follows that link. For a reasonable number of overlapping links (no more than the maximum number that can be listed simultaneously in the link name list) the user can follow a link with one keypress and one click of the mouse. If the user wants to use the mouse only, three mouse clicks are required. For the majority of cases, browsing is very easy. If the number of overlapping links for any given point in the document is very large, the user must scroll through the list of links and choose one. Unfortunately, this extra work is probably unavoidable.

3.2 Window management

An important goal in our interface design was to avoid screen clutter. Like most X Windows applications, Htext takes advantage of the ability to stack and iconify windows already inherent in X. If a user follows a link to a document which is displayed in an existing window, that window is raised. If a user links to a document which is in an iconified window, that window is deiconified.

A less trivial way to help users avoid screen clutter involves managing the creation of windows and the replacement of documents within existing windows. Window management should be flexible, since users may want to keep different numbers of windows on the screen depending on their tasks. If the user is searching through many hierarchically-linked texts for some specific piece of information, she may want to operate in only one window, with destination documents replacing their source documents in that window. If she is reading a paper and occasionally checking references or annotations, she may want at least two windows—one of which always contains the central paper while the other contains side notes. For this purpose, opening many windows would clutter the screen with little benefit. Still other times, she may want to have many windows open at once.

Htext allows the user to control window creation and document replacement. The user may *lock* a window by clicking on the lock button in the upper right-hand corner of the window (see Figure 1), indicating that she does not want the contents of that window replaced. The system maintains a count of the number of unlocked windows currently open. If this number exceeds a user-specified maximum, a new document brought into the system will replace a document in an already existing window (the one which contains the "oldest" document currently displayed). If it exceeds a second

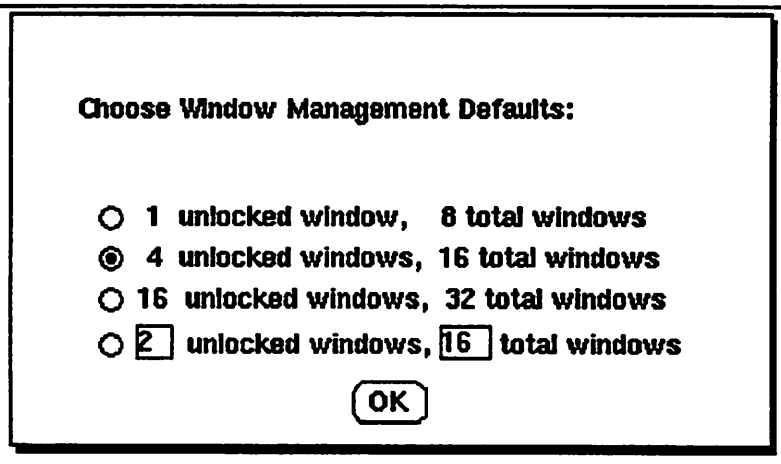


Figure 3: The **Set Window Defaults** dialog box.

maximum, and all available windows are locked, windows that have been locked will have their documents replaced by new ones. Htext provides the dialog box shown in Figure3 with several defaults and an option for setting the maximum number of locked and unlocked windows to arbitrary user-determined values.

Thus, for instance, if a user wants to use only one window, with all link destination documents replacing source documents in that window, she may set the maximum number of unlocked windows to one and leave the window unlocked. This might be useful for casual browsing through a large hypertext document. If at some point the user wants to keep the contents of the window while following a link to another window, she could lock the window, then follow a link, which would create another (unlocked) window. At this point, the user could follow multiple branches from the locked window. At each branch followed, the contents of the second window would be replaced. This type of browsing might be very useful for examining source code distributed over several files, with documentation for all of the source code in a single file, for example.

For other applications, the user might choose to have four windows open simultaneously, without locking any of them. She could then read or browse through a hyperdocument, switching from window to window while keeping a visual trail of the documents she had examined most recently. For still other applications, she could choose to have a very large number of unlocked windows open simultaneously, iconifying windows herself to avoid screen clutter.

In addition to preventing window clutter, window replacement prevents the user from having to continually place windows on the screen. With this management scheme, each window need only be placed once. Users of the system have found these features helpful.

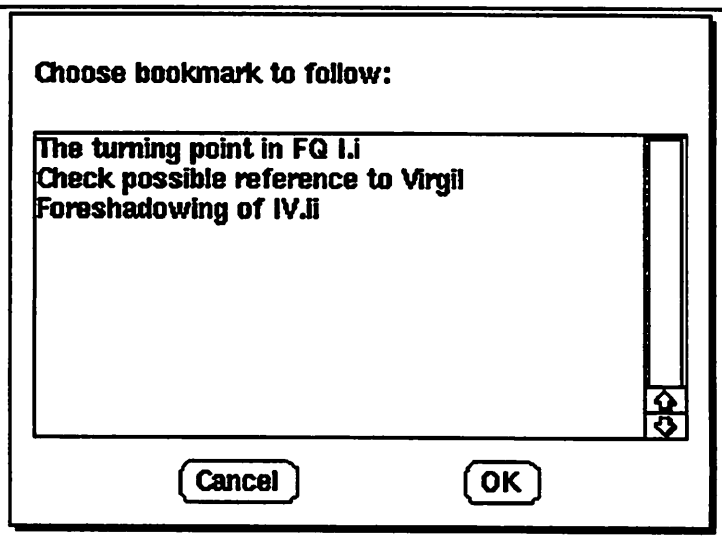


Figure 4: The **Use Bookmarks** dialog box

3.3 Bookmarks and history

One of the major problems associated with using hypermedia systems is disorientation: Users can easily become lost in complex networks of information. Htext provides two simple navigation and orientation aids: bookmarks and a history mechanism. The bookmarks we use are similar in spirit to a feature of the Symbolics Document Examiner [7]. The interface for creating bookmarks in Htext is similar to the interface for creating links. Users specify a region of text or an existing link region as a bookmark and give it a name; they may subsequently visit one of these bookmarks by selecting the **Use Bookmark** menu option. The history mechanism, while simple, is also useful. **Use History** brings up a list of all the documents visited in the current session for the user to choose from. Neither bookmarks nor history are saved between sessions. Figures 4 and 5 illustrate these features.

3.4 Link creation and editing

For convenience, Htext allows users to specify links between multiple source and destination regions all at once. The need for this feature arises often. For example, if one section of a paper is referenced many times in another paper, the user may want to link all of those references to that section. Without the ability to create multiple links at once, the user would have to waste time reselecting the destination region for each new link.

Htext's link creation interface allows users to select any number of regions of text (or existing link regions) in any order and make them into source and destination regions. Figure 6 shows an overview of this process. The user highlights the desired

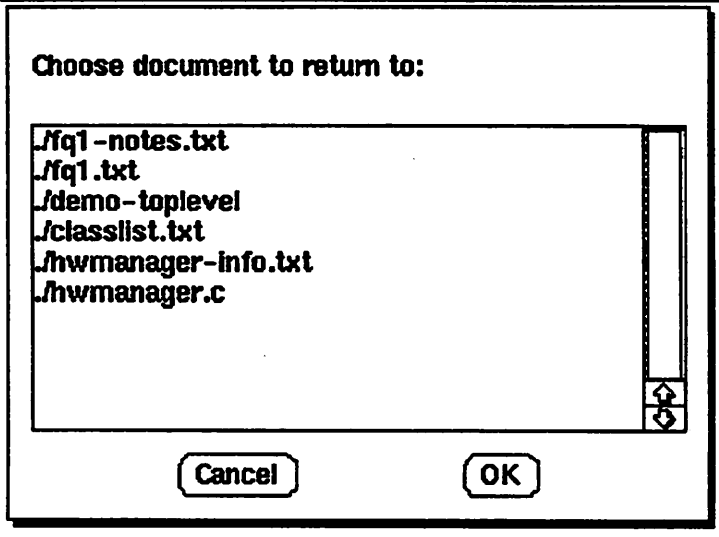


Figure 5: The **Use History** dialog box

text or selects a link from an existing region,[2] then chooses one of the menu commands **Add Link Source** or **Add Link Destination**. To create the actual links between each pair of source and destination regions, the user selects the **Create Links** menu option. Users can delete and rename links by selecting a link using the browser (i.e. by moving the mouse over the source or destination region and selecting the link with the keyboard or mouse) and then selecting the appropriate option from the **Links** menu.

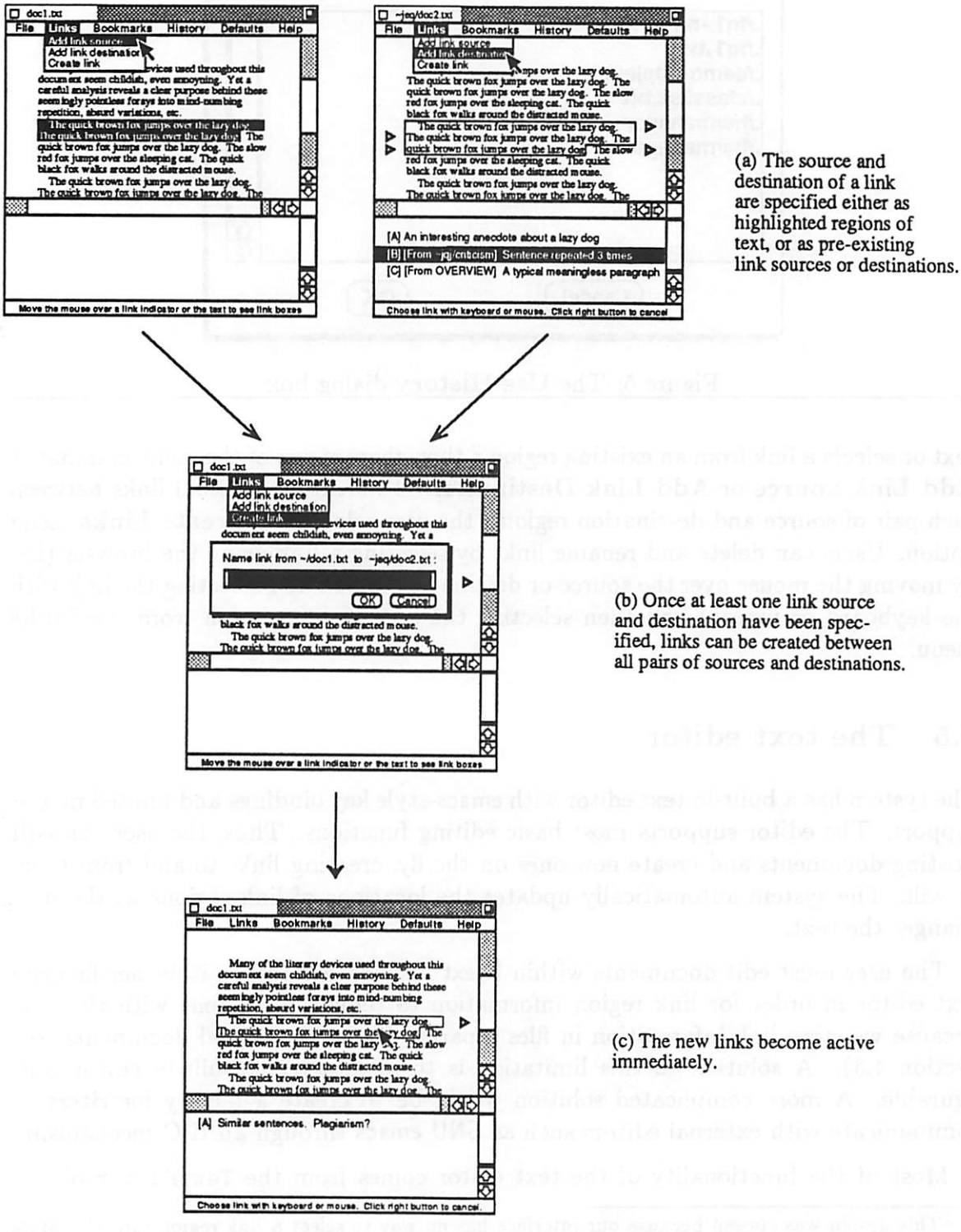
3.5 The text editor

The system has a built-in text editor with emacs-style key bindings and limited mouse support. The editor supports most basic editing functions. Thus, the user can edit existing documents and create new ones on the fly, creating links to and from them at will. The system automatically updates the locations of link regions as the user changes the text.

The user must edit documents within Htext itself rather than within her favorite text editor in order for link region information to remain consistent with the text because we store link information in files separate from the original documents (see Section 4.3). A solution to this limitation is to make Htext's built-in editor configurable. A more complicated solution would be to create a facility for Htext to communicate with external editors such as GNU emacs through an IPC mechanism.

Most of the functionality of the text editor comes from the `TextEditor` object

[2]This design was chosen because our interface has no way to select a link region directly. Since link regions may overlap, it seemed just as easy to use a link from a region to select that region for this purpose.



(a) The source and destination of a link are specified either as highlighted regions of text, or as pre-existing link sources or destinations.

(b) Once at least one link source and destination have been specified, links can be created between all pairs of sources and destinations.

(c) The new links become active immediately.

Figure 6: The link creation process

that is built into the InterViews toolkit. We have not attempted to customize the editor and its associated text display routines, and as a result have encountered some performance problems. Because we store link information in separate link files rather than in the document file itself, the system must update the link file after every block of editing operations. In the future, we may modify the text editor so that when a file is edited, link information is stored invisibly within the text, and updated automatically with every keypress.

4 Implementation

Htext is implemented in the C++ programming language, and runs under X Windows [6] using the InterViews toolkit [4]. We chose InterViews because its object-oriented design provided many high-level interface features (e.g. menus, buttons, and the text editor) that we were able to use "off the shelf" without having to write widget code. While there are some inefficiencies in InterViews, the toolkit's power made coding the interface relatively straightforward. Htext's source code is currently approximately 5400 lines long.

In this section, we discuss some aspects of the implementation of Htext. Section 4.1 briefly describes how we implemented mouse-sensitive link regions in Htext. Section 4.2 describes the general structure of the program. Section 4.3 describes the way in which we achieved file-transparency.

4.1 Mouse-sensitive link regions

Because X Windows and InterViews do not support "intersecting" windows (i.e., overlapping windows that all receive messages about events occurring in the overlap region), Htext creates a separate window in the X server for each region of text that is contained in a particular combination of link regions. For example, in Figure 7, three windows are created—one for the portion of link region 1 that does not overlap link region 2, one for the overlapping region, and one for the portion of link region 2 that does not overlap link region 1.[3] Each window is transparent, and receives events when the mouse cursor is moved into or out of it or when the mouse button is depressed in it. When such an event occurs, the X server sends messages to Htext informing it of the event and of the pertinent link region window. We have encountered no performance problems with this scheme, even when there are many link regions (and thus many windows).

[3]Actually, since X Windows and InterViews only support rectangular windows, regions are further subdivided into up to three windows—one containing the partial line at the beginning of the region (if any), one containing the partial line at the end of the region (if any), and one containing the lines in between.

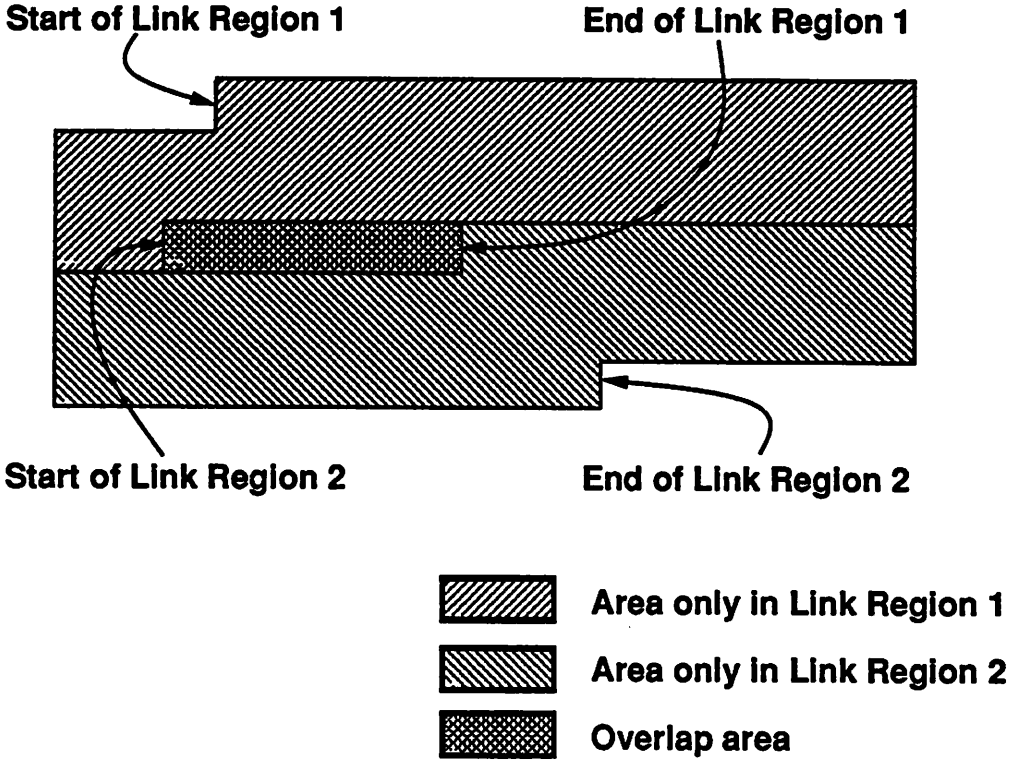


Figure 7: Overlapping link regions

4.2 Object classes and abstractions

Because InterViews is written in C++, an object-oriented programming language, our interface elements are necessarily C++ objects. Most of our non-interface data structures also fit into the object-oriented scheme quite well. Using objects and methods rather than global structures and functions helps keep our code organized, which facilitates our debugging and updating the system.

To insure consistency between the various types of documents that Htext may handle in the future and reduce redundancy in the code for these document types, the system uses instances of the same object, `HyperWindow`, to provide interfaces for each Htext window on the screen, regardless of the type of document contained in the window. By itself, this `HyperWindow` contains a menu bar at the top of the window, and a string browser and message line near the bottom of the window. It also contains space for an object of type `HyperDocument`. The `HyperWindow` handles menu choices and displays link names in the link name list.

Code that is common to all types of documents is part of the class `HyperDocument`. The functions that handle individual types of documents are implemented as subclasses of `HyperDocument`. Currently, only one such subclass exists—`HyperTextDocument`. We discuss how other document types may be derived from the `HyperDocument` class in section 5.2.1.

While `HyperWindows` handle everything that happens local to a window, a supervisor object, `HyperManager`, is responsible for functions that are not specific to particular windows, or that involve more than one window. These functions include: (1) opening documents, (2) following links, (3) managing link creation, renaming, and deletion, and (4) keeping track of bookmarks and history. At a lower level, `HyperManager` is also responsible for reading events from the system's event queue and dispatching the events to their target InterViews interactors.

Several other objects assist the `HyperManager` in its tasks. The `HyperWindowManager` class has methods to find the window into which a given document has been loaded and to check the number of locked and unlocked windows during the opening of a new document. The `LinkFileDB` class has methods to retrieve and update information in link files. Similarly, the `DocumentDB` class has methods to process information in the document database files, which keep track of the pairings between documents and their link files. These classes handle reading and writing to disk, sometimes calling document type-specific routines to preserve compatibility with future updates to the system.

4.3 Link information files

As mentioned in the introduction, file-transparency is an important feature of Htext. Htext achieves this transparency by storing link information in link information files

(hereafter referred to as *linkfiles*) that are separate from the original documents. Furthermore, it keeps track of the pairings between linkfiles and associated documents in a document database file. This way, when the user opens a document, all links to and from the document are loaded automatically, even though the linkfile may be stored at an entirely different location in the directory structure.

Within each individual linkfile, Htext stores link region objects separately from link objects. Rather than storing the start and end points of a link's source and destination regions in the link object, the system stores the indices of the link region objects (see Figure 8). This is primarily useful when the user edits a document. For example, if the user inserts text into the dotted link region shown in the DestFile window in Figure 8, the boundaries of that link region and all regions after it must be changed in the DestFileLinkDB, but none of the information in the SourceFileLinkDB needs to be changed, since the link from the SourceFile points only to the index of the destination region. Furthermore, if the system is expanded in the future to include other types of documents, the link file format for text-only documents need not change, since the links from a given file only include pointers to regions in the linked document. For example, a paragraph in a text-only document could easily be linked to a rectangular region in a bitmap in another file.

5 Discussion

Section 5.1 discusses the current status of Htext's implementation; Section 5.2 mentions possible extensions to the system.

5.1 Htext's current status

We have used the system to build several sample hyperdocuments in different domains. The system's interface design is flexible and powerful enough to support their creation while providing a facile browsing mechanism. Htext's window management scheme has proven extremely useful in allowing us to focus our attention on key documents while rapidly traversing links between many others.

At this stage we have not performed any human factors experiments. However, several people have used the system and commented favorably about its window management and link browsing interfaces.

5.2 Possible extensions

Though we feel it is important to keep Htext simple and conceptually clean, there are a number of ways the current system could be extended without detracting from its clarity of design.

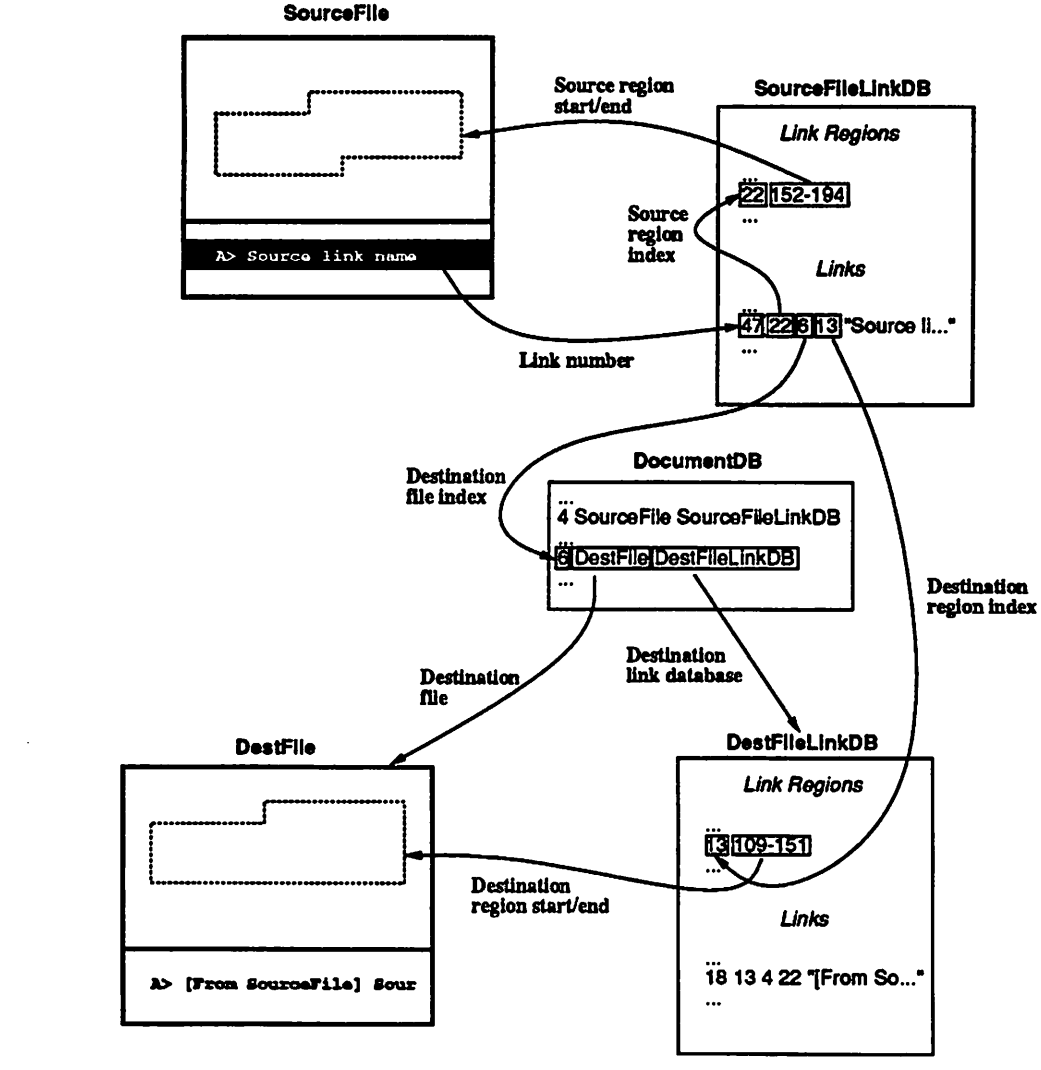


Figure 8: Link storage

5.2.1 Other types of documents

As mentioned in section 4.2, the system contains provisions for incorporating code to handle non-plaintext documents by deriving subclasses from class `HyperDocument`. This class contains a set of virtual methods that its subclasses should implement in order to hook into the rest of Htext. Each document subclass is responsible for formatting and drawing its contents and managing events within its display area (link region entrance, exit, and mouse-button events in particular). For instance, `HyperTextDocument`, the only subclass we have implemented, creates link region windows and calls text editor functions. A graphical document handler might allow arbitrarily-shaped link regions and include a painting or object-drawing module; a sound-sample handler might display a two-dimensional waveform and have a link region be some segment of that waveform. The system currently has no provision for allowing subclasses to attach menus to `HyperWindows` (if, for example, the programmer would like the document-handler's editor to be menu-driven), but this feature would be relatively easy to implement.

5.2.2 Multiuser support

Although Htext does not currently provide explicit support for multiple users, the link information storage system can easily be extended to accomodate them. More specifically, multiple linkfiles could be stored for each document—for example, one public linkfile and numerous private linkfiles—and a separate database could be kept for each public or private set of linkfiles. While this method does not address problems of concurrent file access, it would conveniently handle public and private links.

5.2.3 Webs

In a heavily-used hypertext system with a large number of links made by various people for various purposes, activating all the links could severely confuse users. For example, someone browsing through hierarchically-structured documentation might be distracted by links made by other people to this documentation for their own use. Some of these problems, but not all of them, could be solved with the public and private linkfiles discussed in the last section. A more general solution is the use of *webs*, as described in [1].

The basic concept of webs is that links with similar purposes (or made by the same person or in the same general time period) can be grouped together. Users can exclude certain webs from their views, so they only see the links in which they are interested. We believe it is best to activate links from all webs by default, then give the user the option of excluding links from certain webs (or, conversely, activating only links from certain webs). Our back-end code and linkfile format have partial support for webs, but at this stage we have not yet implemented all of the necessary

code and interface elements.

5.2.4 Contexts

Sometimes a link to a specific section of a document is not sufficient for specifying the area of interest in the linked document. Surrounding the linked area of a document, there may be a section of the document that is less valuable but still important, or a section that places the linked area in an appropriate context. Also, there may be sections of the linked document that are not relevant at all and that the user should not have to see when following a link. For example, a link might point to two sentences in one e-mail message in a file of assorted e-mail messages; the user would probably not be interested in any messages but the relevant one, but would probably be interested in the entire text of the relevant one.

We plan to add a new entity, the *context region*, to solve this problem. If the user follows a link that has a context region associated with it, the system will limit the visible region of the destination file to the context region, which may be larger than the link's actual destination region.

Acknowledgments

Jamie Zawinski stood still while we bounced ideas off of him in the early stages. Morrisa Sherman provided us with the annotated version of *The Faerie Queene* that we used in testing the system. James Joyce and Thomas Pynchon were major inspirations for this project.

References

[1] J. Conklin, 'Hypertext: An Introduction and Survey,' *IEEE Computer*, September 1987.

[2] D. C. Engelbart and W. K. English, 'A Research Center for Augmenting Human Intellect,' *AFIPS Conf. Proc.*, Vol. 33, Part 1, The Thompson Book Company, Washington, D.C., 1968.

[3] F. G. Halasz, *et al.*, 'NoteCards in a Nutshell,' *Proc. of the ACM CHI+GI 1987 Conf.*, Toronto, Canada, April 1987.

[4] M. A. Linton, *et al.*, 'Composing User Interfaces with InterViews,' *IEEE Computer*, February 1989.

[5] N. Meyrowitz, 'Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework,' *Proc. OOPSLA '86*, Portland, OR, September 1986.

[6] R. W. Scheifler and J. Gettys, 'The X Window System,' *ACM Trans. on Graphics*, Vol. 5, No. 2, April 1986.

[7] J. H. Walker, 'The Document Examiner,' *SIGGRAPH Video Review*, Edited compilation from *CHI '85: Human Factors in Computing Systems*, 1985.