A Hypertext Display Component For A Graphical User Interface Development Environment

Richard J. Love
Decision and Information Sciences Division
Argonne National Laboratory

Abstract

Hypertext is often used in the World Wide Web and in application help tools, but it is certainly capable of much more. If it was available to application programmers as another graphical user interface component, like a button or an image, a wider range of use could be enabled. The Hypertext Display System (HDS), provides a hypertext component which can then be incorporated into a graphical user interface (GUI) development environment.

The HDS consists of a hypertext display component, called the HyperDisplay, and a test-bed in the form of a local HTML file browser. Its distinctive characteristics are (1) it was developed with an object-oriented design, using C++, for the Motif X toolkit, (2) it encapsulates the hypertext display capability in the reusable HyperDisplay object, so that it can be easily included in other applications, and (3) the HyperDisplay object is designed with portability in mind, so it can be ported to additional systems. This paper describes the HDS and the HyperDisplay component with: an introduction and design overview, including the class subsystems; a high-level view of their implementation; and a discussion of future directions.

Keywords

Hypertext, Component, GUI, HTML, C++, Motif, Object Oriented Design

1 Introduction

Many people are astonished when they first use a hypertext browser. By simply using mouse clicks, the user is able to navigate through a complex network of documents and control the order in which the documents are viewed. There are many other types of hypertext applications that could be created; browsers just scratch the surface. Possible applications could include: hypertext editors and writing tools, automated hypertext document constructors, digital library tools, and tools that facilitate collaborative workgroups over hypertext networks. These would broaden the use of hypertext and make it accessible to even more people, but there needs to be an efficient way to create these applications. More specifically, there needs to be a way to incorporate hypertext capabilities into a wide range of applications. The Hypertext Display System (HDS)[1] addresses this need.

[1] This work was not funded through Argonne National Laboratory.

2 Design Overview

The HDS was developed using an object-oriented design (OOD) that stresses generality, functionality, and portability. The design should comfortably allow the addition of extended functionality, such as: reading various file formats, writing out various formats, hypertext editing, and hypermedia display. The HDS should run on any UNIX system with X Windows, Motif, and a graphical window manager. It has been tested on Sun Workstations (SPARCstations 2 and 5), running Solaris (2.3 and 2.4). The class subsystems of the HDS are discussed in detail later, but generally, all displayable objects consist of basic X Toolkit widgets that have been wrapped by C++ classes.[2] No special tool packages were used.

There are several reasons why an OOD was chosen for the HDS. This paper will not offer arguments for the advantages of the object-oriented software life cycle over the traditional life cycle or for the advantages of an object-oriented programming (OOP) language over a procedural language. Since the HyperDisplay will be incorporated into the AIE, and since the AIE was developed with an OOD using C++, it was natural to also use these for the HDS. Also, some HTML elements already encapsulate objects (e.g. paragraphs) and the rest naturally lend themselves towards a hierarchy of groups of element types.

The Motif graphical user interface was chosen because it has become the de facto standard on UNIX platforms. All of the HDS's Motif code is written in the C language, because X Windows provides a C language client interface, and C interfaces well with C++.

In order to make it easier to port the HDS's hypertext display conponent (called the HyperDisplay) to another GUI, such as Microsoft Windows, all the Motif code is isolated in a separate section, called the windowing system dependent code. The HyperDisplay's C++ classes are designed to be independent of the GUI. Using this strategy, porting the HyperDisplay to another GUI would require porting only about 25% of the code.[3]

The HDS reads HTML level 2 formatted files, reachable in the local file structure. The HTML level 2 format is more strictly structured than the level 1 and 0 formats; making it easier to parse.

3 Approach

The HDS consists of the HyperDisplay incorporated into a local HTML file browser [5]. The browser allows users to view hypertext documents that reside in the local file-space and that are in the HyperText Markup Language (HTML) level 2 interchange format. The HyperDisplay uses a scrollable region to display the hypertext elements in the HTML documents and provides basic navigational capabilities.

[2] The X Toolkit is the combination of the X Toolkit Intrinsics and the Motif widget set. The X Toolkit Intrinsics are based on the X library, the C programming interface to Version 11 of the X Window System.

[3] This strategy has been used successfully in the AIE.

The following is a typical scenario, describing the use of the HDS. After starting the HDS, the user chooses the Open command from the File menu, and then selects a document from a selection dialog. The HDS tells the HyperDisplay what document to open. The HyperDisplay starts parsing the HTML elements, creating the appropriate hypertext objects referenced in the document, and then displays the objects. Figure 2.1 shows a sample HTML document (listed in Example 2.1) as displayed in the HDS. Any text marked as a link to another document (i.e. an anchor) is underlined. The mouse cursor changes when it is within the boundaries of the link, and the name of the link's destination document is shown at the bottom of the HDS window. When a link is selected, the referenced document is loaded in the same manner as the current document and replaces the current document in the text region. The displaced document is added to a memory resident linear list of documents traversed via the links, called the Page History list. Through "Back," "Forward," and "Home" menu items, the user can navigate documents by replacing the currently displayed document with the one before or after it, respectively.

Work is currently being done to incorporate the HyperDisplay into the Application Interface Engine (AIE), an existing object-oriented development environment developed at Argonne National Laboratory [2][7]. The AIE allows rapid prototyping of graphically oriented applications by shielding the user from the low-level programming normally required in GUI development. Instead, the user programs in the AIE's object-oriented language, which can then be compiled for the desired target operating system. After the HyperDisplay is added, an AIE programmer will be able to create applications that include hypertext components. These applications could be entirely hypertext based, simply include a hypertext component (e.g. for supplying helpful information to the user), or they could use hypertext in a new way. One example of a program that could be created is a writing tool that allows authors to keep all of their notes organized and linked together.

Hypertext Features

In the HDS, nodes are called pages and consist of one HTML document. The reason is that, like most browsers, the HDS has a scrollable region that can display any size HTML document.

The HDS displays anchors as underlined text. When the user moves the mouse cursor over an anchor, the cursor changes to a pointing hand and the anchor's destination, the path of the linked document, is displayed in the status label below the hypertext display area (see Figure 2.1).

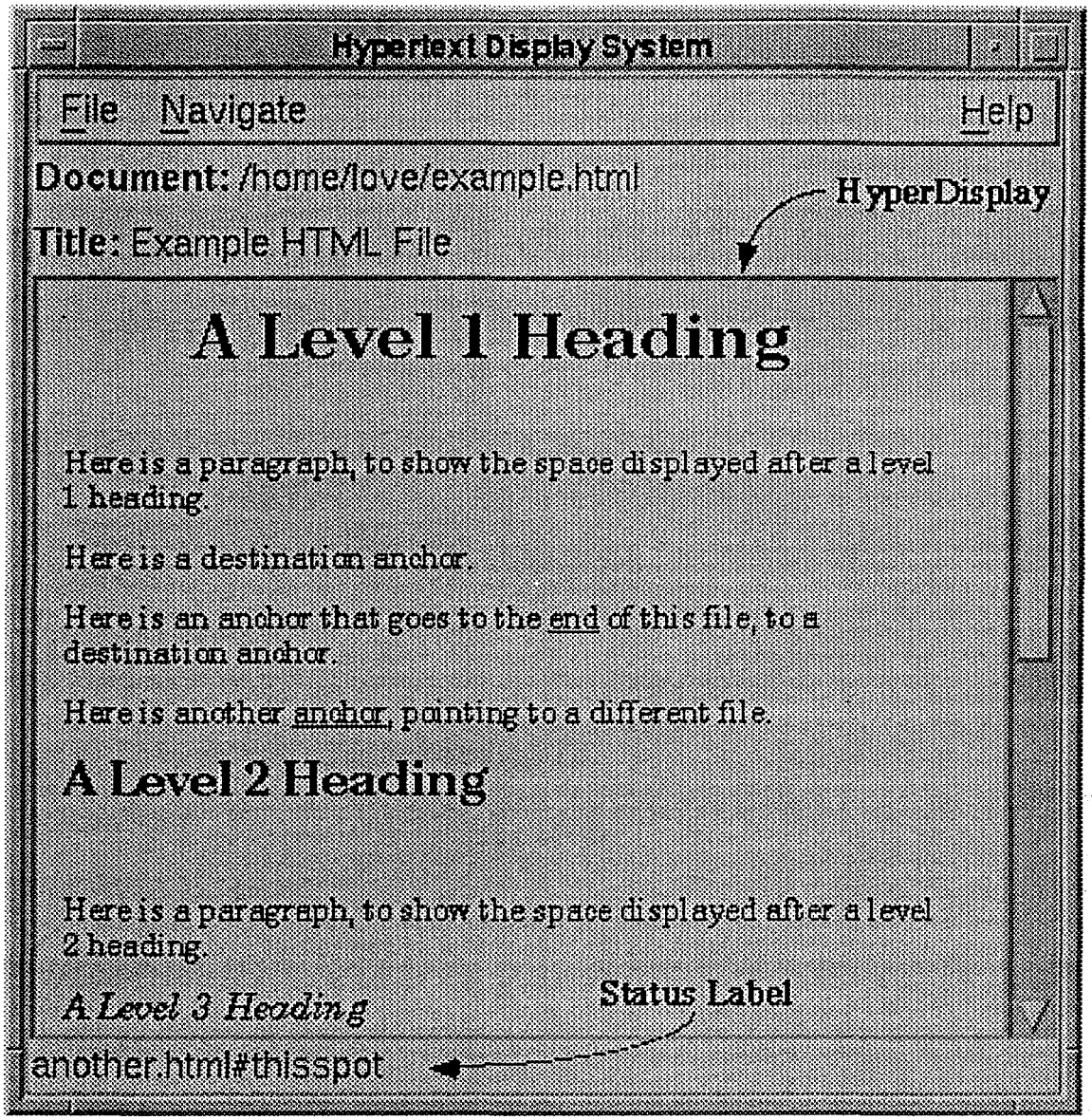

Figure 2.1 The HDS with Example 2.1 loaded and the mouse cursor over an anchor

Example 2.1 A sample HTML document

```
<!doctype HTML public "-//W3O//DTD W3 HTML 2.0//EN">
<html>
<head><title>Example HTML File</title>
</head>
<body>

<h1>A Level 1 Heading</h1>

<p>Here is a paragraph, to show the space displayed after a level 1
heading.</p>

<p>Here is a <a name="thetop">destination</a> anchor.</p>

<p>Here is an anchor that goes to the <a href = "#theend">end</a> of
```

```
this file, to a destination anchor.</p>

<p>Here is another <a href="another.html#thisspot">anchor</a>,
pointing to a different file.</p>

<h2>A Level 2 Heading</h2>
<p>Here is a paragraph, to show the space displayed after a level 2
heading.</p>

<h3>A Level 3 Heading</h3>
<p>Here is a paragraph, to show the space displayed after a level 3
heading.</p>

<h4>A Level 4 Heading</h4>
<p>Here is a paragraph, to show the space displayed after a level 4
heading.</p>

<h5>A Level 5 Heading</h5>
<p>Here is a paragraph, to show the space displayed after a level 5
heading.</p>

<h6>A Level 6 Heading</h6>
<p>Here is an anchor that goes to the <a href="#thetop">beginning</a>
of this file.</p>

<p>Here is a <a name="theend">destination</a> anchor.</p>

</body>
</html>
```

There are other ways to navigate between HTML documents in the HDS, besides using anchors. To load a new HTML document, the Open command in the File menu can be used. A file selection dialog box displays, allowing the user to choose a document from the local file structure. Once selected, the document loads into the HyperDisplay, replacing the previous page. The previous page is still saved in a linear list, in memory. Back, Forward, and Home commands (HDS menu items in the Navigate menu) can be used to reload a page that has been saved in the linear list. The Back command reloads the previous page. The Forward command reloads the next page (possible only after a Back command is used). The Home command reloads the first document that was displayed.

Class Subsystems

The classes included in the HDS are grouped into four subsystems: HyperDisplay, which encapsulates the hypertext and HTML capabilities; HTMLElement (a subsystem of HyperDisplay); test-bed; and support. These subsystems are shown in Figures 2.2 and 2.4. The responsibilities of the HyperDisplay and test-bed subsystems are listed in Figure 2.3.

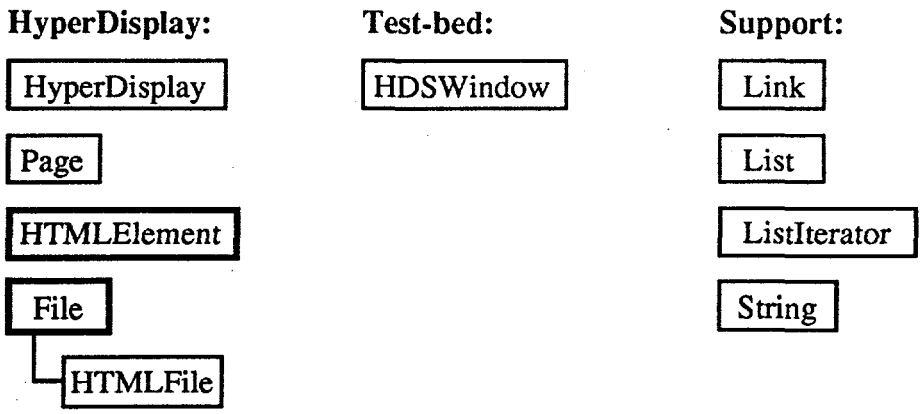

Figure 2.2 HDS class subsystems.

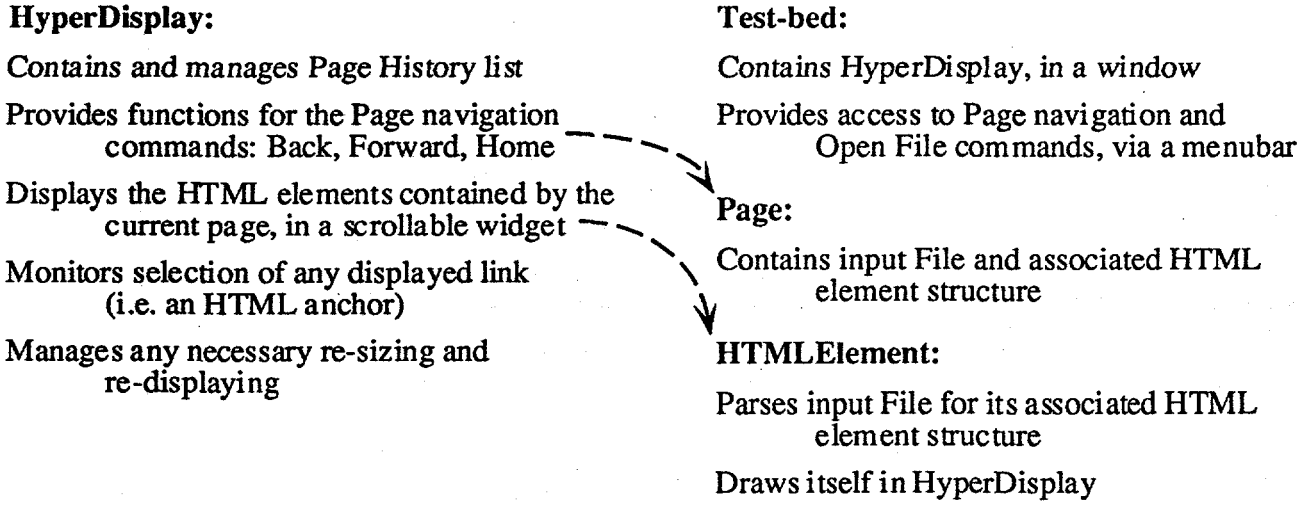

Figure 2.3 Responsibilities of the HyperDisplay and test-bed subsystems.

The HyperDisplay and Page classes, along with the HTMLElement subsystem, encapsulate the hypertext functionalities. Together, they compose the hypertext GUI component. The HyperDisplay handles hypertext display, navigation, and Page History management. A Page manages an associated File and the hypertext objects described in the File. The File class handles the reading and lexical analysis of input files. The subclassed HTMLFile class is specifically designed for HTML files.

The only actual class in the test-bed subsystem is the HDSWindow, which handles the display of the HDS window (containing the HyperDisplay), menubar, and status label (see Figure 2.1). The test-bed could be expanded in the future, causing more classes to be added to this subsystem.

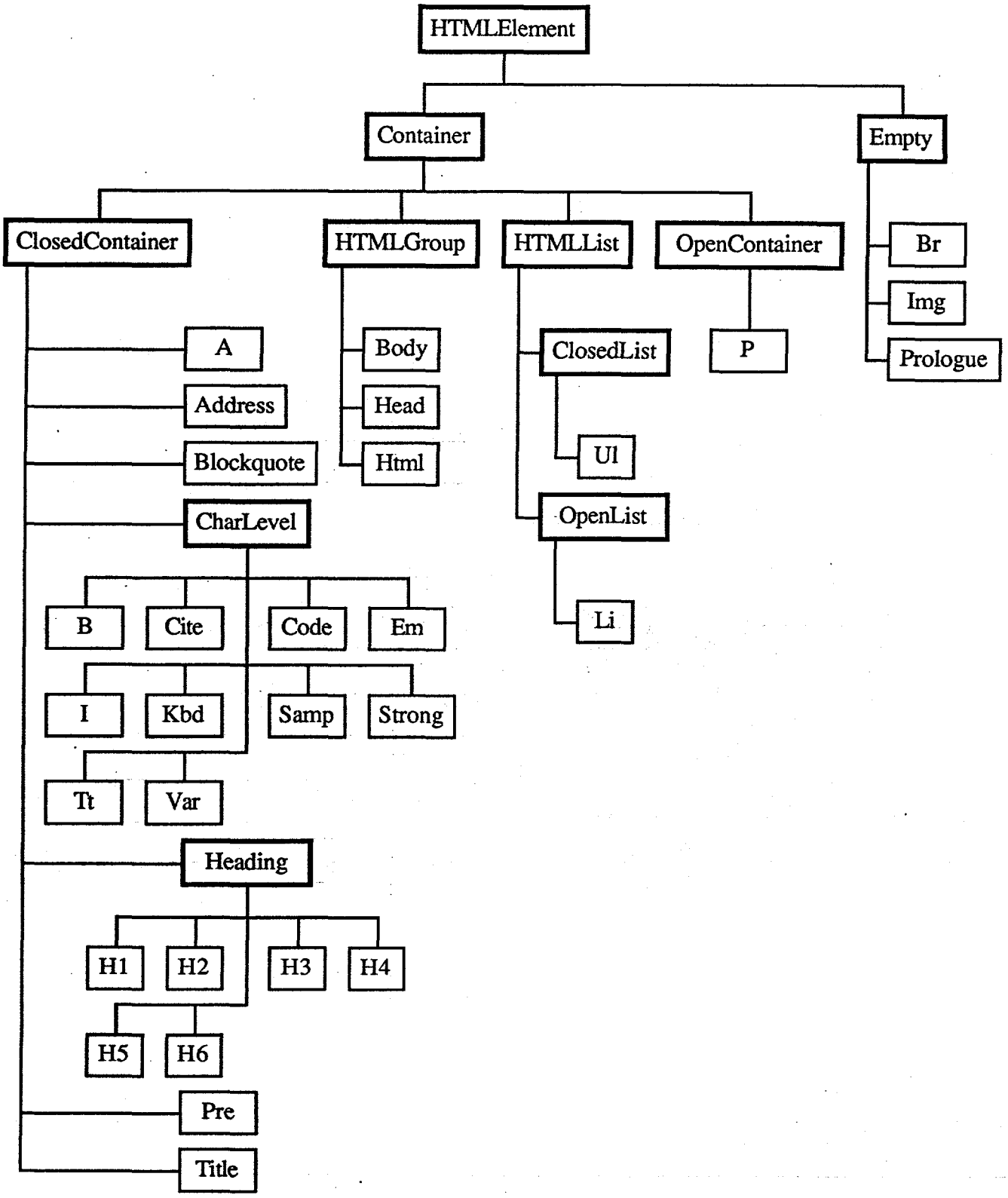

Figure 2.4 HTMLElement class subsystem

The Link, List, ListIterator, and String classes are fairly standard supporting classes. The List and Link classes are used to implement a linked list. A ListIterator is used to traverse the linked list. The String class is a version of the basic C++ character array class.

The HTML elements were analyzed and grouped by functionality. Then abstract classes were subclassed from HTMLElement, and each group of elements was subclassed from the abstract classes. A Container element contains text or a list of elements. A ClosedContainer element has tags that indicate both the start and end of that element's text. An OpenContainer on the other hand, may only have one tag at the beginning or end. An HTMLGroup element does not contain text, but contains a list of elements. For example, the Html element can only contain optional Head and Body elements. The HTMLList element can contain both text and other elements. An Empty element does not contain text to be displayed, but supplies information for the browser. For example, the Br element tells the browser to start a new line.

3 HDS Implementation

Currently, the HyperDisplay displays most of the text elements of an HTML level 2 document. Plans do specify, however, that development will continue until the HyperDisplay is able to recognize all HTML elements, read all HTML levels, display images as well as text, and handle HTML forms.

Specific descriptions of the HDS's implementation (e.g. the algorithms and Motif code) are not described here, but a high level view of the core functionality is described. These functionalities include: input/output handling, parsing the HTML file, drawing the HTML page, and hypertext features.

Input/Output

When the user selects a file to be opened, the HyperDisplay creates a Page object, to manage both the file itself and the hypertext objects described in the file. Thus, the Page object is in charge of parsing the file. For managing the file, the Page object contains a HTMLFile object. The HTMLFile object handles the actual inputting of local HTML files. The HTMLFile class, though, acts as more than just an expanded C FILE type. It also handles the lexical analysis chores, while reading in an HTML file. A HTMLFile can also write text out to an ASCII file (the HDS does this for an informational log file), but it does not presently write the HTML format.

When the file is first opened, the HTMLFile object reads in a 512 byte section of the HTML file, placing it into a buffer. The HTMLSystem object then tells the Page to begin parsing the HTML file to get the HTML elements. To do this, the Page utilizes the HTMLFile's lexical analysis member functions. At the beginning of the file, the Page object looks for either a Prologue or the start of a tag token ('<').

Whenever the Page finishes parsing the section of the file in the buffer, a new 512 byte section is loaded. If a token straddles the boundary between sections, the HTMLFile will read in the first part of the token, load in the new section, and read the rest of the token. The HTMLFile object manages a pointer to the current position within the buffer, and has member functions that can be used to move forward and backward, one character at a time.

HTMLFile is subclassed from the File class strictly for generality. If the HDS is expanded to read in another file format, a new class could be created as a sibling to HTMLFile.

Parsing

Every HTML file is supposed to have a prologue element at the very beginning of the file. When the HDS parses a file without a prologue element, it writes out a warning message to the log file, but continues to parse. Comments can appear anywhere in an HTML file, but are ignored by the lexical analysis functions [1].

At this point, the Page determines what HTML element is at the beginning of the file. If it is the prologue, then a Prologue object is created. If not, then an Html object is created. The newly created object then takes over the parsing tasks. When the Prologue completes its parsing, it returns control to the Page, which creates an Html object. The Html object then begins parsing, looking for the Head and Body elements.

This process, of parsing until a new element is found, creating an appropriate new object, and then handing over the parsing control to the new object, is continued for the entire file. This is one of the advantages of object-orientation; tasks such as parsing can be performed in a recursive manner.

If an element contains other elements as well as text, a marker is inserted at each contained element's location. An object is then created for the contained element, which parses itself as described above.

Drawing

Once a file is completely parsed, the HTMLSystem tells the Page to draw itself in the HyperDisplay. The Page tells the Html to draw itself, which tells its associated Body to draw itself. This is similar to the parsing functionality. The Body object mainly consists of a list of contained elements, and tells each to draw themselves. While looping through the elements, the Body keeps track of each one's height, and provides each succeeding element with a position at which to begin drawing.

These elements may also contain other elements, so they use a similar looping scheme to draw themselves. Most of the Container classes may contain both text and other elements, unlike the Body class, which does not contain any text of its own. As a result, the algorithm used in the draw functions of classes such as P(aragraph) and A(nchor) is more complicated.

Objects of these classes have markers within their text indicating the location of each contained element. During drawing, the object will first draw the segment of text before any contained element. Upon encountering a marker, the object tells the appropriate element to draw itself, supplying it with a starting position. When the contained element is finished, the object then continues, looping until all segments and contained elements are drawn.

The HTMLElement class' draw functions are separated from the parse functions for a good reason; the draw functions can then be used again without necessitating another parse. This is utilized by the HDS when one of the navigation commands is used to go to another page in the history list, when the HyperDisplay is cleared and the selected Page object is simply told to draw itself again. It is also used when the user changes the size of the HyperDisplay (through the window manager). When this occurs, the HyperDisplay tells the currently displayed Page to draw itself again.

There are several ways that the HDS could draw each text segment. Since it has been implemented for the X Windows system, it could use the Xlib string drawing functions.Since the HDS uses the Motif widget set, it could also use its versions of the string drawing functions. These are similar to the Xlib routines, but do a little more of the work for the programmer. One of the routines will actually draw a string and underline a sub-string within it, useful for anchors.

The HDS, though, takes advantage of the Motif widget set and creates a Label widget for every segment. The Label widgets have additional capabilities not needed by the HDS, so they have additional memory overhead, but they can also do more work for the programmer. They maintain their own appearance variables, draw themselves, and redraw themselves when necessary.

Hypertext Features

As may have been realized from the previous few sections, the Page class embodies the hypertext concept of the node. A Page object contains an HTML document; it holds a pointer to the file and it holds all of the HTML objects within an Html object. To make the contents of a Page accessible to the user, they are displayed in a HyperDisplay object.

The HyperDisplay class is built around Motif's DrawingArea widget, which can display text, graphics, and other GUI components. This allows the HyperDisplay to display all of the HTML elements (and allows the HyperDisplay to be expanded to handle in-line images and the HTML form element).

The embodiment of the hypertext link concept is provided by the HTML Anchor element. There are two types of anchors in HTML, departure anchors and destination anchors. Thus the HyperDisplay currently supports these two types of links. Departure anchors are underlined when drawn, making them visible to the user. When selected, a corresponding page is loaded into the HyperDisplay. A destination anchor is invisible to the user, but is managed by the HyperDisplay for navigational purposes.

The Page History list is another key hypertext feature implemented by the HyperDisplay. Every time a new HTML file is opened or navigated to, the HyperDisplay creates and adds a Page object to the Page History list, using available memory. This list is maintained as an object-oriented linked list in which each link is a Page object. The HyperDisplay also holds pointers to the current and last Pages in the history list. These pointers are utilized when the user activates any of the history list navigation commands: Back, Forward, and Home, in the HDS Navigation menu.

4 Future Directions

Before the HyperDisplay is added to the AIE language, one major enhancement will be made; all of the text elements of HTML level 2 will be recognized by the HyperDisplay and drawn appropriately, including special characters. The HyperDisplay's existing HTML class structure already includes all such elements, because they had been considered during the design process. The capability to parse HTML level 0 and level 1 files may be added as well. This could be made possible if each HTML class is enhanced so it is able to recognize its own level 0 or level 1 versions. The class for the HTML image element will be enhanced to enable it to display its in-line images. (Presently, it only displays its alternative text). Classes for the HTML form element and its related elements may also be added.

Once the HyperDisplay is added to the AIE, more hypertext navigation tools could be implemented using the AIE's other GUI objects. More complex page history lists could be implemented. Bookmarks, could be created to save hypertext locations so they could be returned to in the future. New types of links could be added, such as a popup link, adding to the HTML's limited two types of anchors. When a popup link is followed, for example, a small window could pop up with the linked text. These new links could have different cursor shapes for their anchors and/or be highlighted in a different way.

5 Conclusion

The HyperDisplay is the basic component of almost every envisioned hypertext system. Some of these systems exist already, and are widely used, such as the Web browser Mosaic. Others are in the early stages of development and/or use, such as the HTML editors SoftQuad HoTMetaL Pro, Arachnid, and BBEdit. Still others are experimental, such as tools for automating the construction of HTML documents, converting common word processing formats to HTML, and facilitating collaborative workgroups over a hypertext network [3]. When available as a component in a GUI development environment such as the AIE, the HyperDisplay fulfills the need for an efficient way to create hypertext applications.

References

The following reference is for an Internet draft found on the World Wide Web. A Universal Resource Locator (URL, sometimes called a Uniform Resource Locator) is an address of the World Wide Web. An Internet draft is a working document valid for a maximum of six months. (Internet Drafts may be updated, replaced, or obsoleted by other documents at any time.)

1. Berners-Lee, T and D. Connolly. *HyperText Markup Language Specification - 2.0*, Working Draft: Version 1.2. URL: http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html.

2. Fuja, R. and M. A. Widing. *Application Interface Engine Language and Object Reference Manual.* Argonne National Laboratory, IL, TM-72, February 1992.

3. Hardin, J. and I. Goldstein. (Eds.) *Proc. The Second International WWW Conference '94: Mosaic and the Web.* (Chicago, IL, 17-20 October 1994).

4. Nielsen, J. *Hypertext & Hypermedia.* Academic Press Professional, Cambridge, MA, 1993.

5. Love, R. *A Hypertext Display System.* North Central College, 1995.

6. Seyer, P. *Understanding Hypertext: Concepts and Applications.* Windcrest Books, 1991.

7. Widing, M.A. et. al. *Application Interface Engine Application Developer's Manual.* Argonne National Laboratory, IL, TM-61, July 1990.

8. Wirfs-Brock, R., B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software.* P T R Prentice-Hall, NJ, 1990.