Experiences in Writing a WYSIWYG Editor for HTML

Nick Williams
Tim Wilkinson
Systems Architecture Research Centre
City University

April 15, 1994

Abstract

It is well known that HTML is not the most intuitive of languages in which to write textual documents. There are constant requests for a WYSIWYG editor which will ease this task. Such an editor provides obvious benefits: ease of use and the guarantee of correct HTML documents. Prospective Web authors are often deterred upon finding they need to learn a new language or editor before they can even start to author a document.

This paper describes a freely available editor, written at City University, for creating HTML documents. The editor is implemented using a WYSIWYG system known as the Andrew User Interface System, a multi-media object-oriented toolkit, of which an overview is given with a description on why it makes a good basis for an HTML editor. The editor provides a complete implementation of HTML and has been used extensively within City University to produce Web documents.

A comparison between other editing systems such as tkWWW and using Microsoft Word is given, highlighting the various benefits and drawbacks of the different systems. The different approaches for providing *semantic* mark-up through a WYSIWYG interface are described; many constructs (for example, lists) cannot be expressed intuitively in a point and click interface. A description is given of the approaches taken in the implementation of our editor, describing some of the more unusual design ideas. Some of the deficiencies of the current implementation are shown and what features should be provided by other authors of WYSIWYG editors, as elicited by feedback from users of our editor, are also presented.

A description is given of the facilities available within the Andrew Toolkit for implementing such things as tables, figures and more complex objects, thus showing how the editor can be simply extended to provide a full implementation of the developing HTML+ standard.

1 Introduction

Currently, an explosion of interest in the World Wide Web is taking place, with more and more hopeful authors appearing every day. Usually however, the sort of people who provide interesting information are not those comfortable with learning several new languages before breakfast, but are rather more sane people who don't like the idea of learning a new language in which to write their documents.

Documents found within the World Wide Web are usually written in the language known as HTML, the Hypertext Markup Language[1]. This language is a structural and semantic markup language, which specifies not how the document should appear, but rather allows the author to annotate a document with comments on its structural content. These two different views are often referred to as physical and logical respectively. A physical style is, for example, a directive to make a section of text bold, or italic. A logical style is an annotation marking a section of text as a quotation, or emphasised; it is a higher level description. The advantage of logical styles is that the decision on how to render the text is taken at the last possible moment when it is presented to the reader, taking into account the limitations of their display. This allows web documents to be viewed on a variety of different hardware capabilities with the minimum of differences. Of course, HTML also provides physical styles as a fallback for when a particular idea cannot be expressed logically in simple HTML.

Web documents do not *have* to be written in HTML. Documents may be made available in plain textual format, imitated formatting (setext), or indeed anything else that might be acceptable to the end client. If necessary, a conversion can always be built into the server which can provide clients with documents in whichever form desired. However, HTML is guaranteed to be understood by all clients. Furthermore, HTML provides a mechanism for specifying hypertext links to other points of the web.

While HTML has obvious benefits, it is also one of the major stumbling blocks to new authors; why should an author have to learn "Yet Another Language?";. The language HTML is often criticised for lack of features, although this has been somewhat fixed by the introduction of the new version of HTML known as HTML+. Unfortunately, at this point in time there are few browsers for that version of the language.

Within the Computer Science department of City University, we have developed a system for editing HTML documents in a WYSIWYG manner, using as its foundation the Andrew Toolkit (ATK), a part of the Andrew User Interface System (AUIS). ATK is a multi-media toolkit which provides many features which are essential in providing a view onto HTML documents, and is discussed in more detail later.

In this paper, we describe our editing system and some of the experiences gained from writing it, reflecting views on HTML and Web documents in general. We hope that this information will be valuable to future developers of similar applications. We also present a brief review of other existing packages for editing HTML, with a discussion of the various merits of each and how the different interfaces tackle the problems outlined in the next section.

2 A discussion of HTML

HTML is a language for logical markup of documents. A major goal of HTML is to compel authors to create documents using only logical markup rather than physical effects. This is similar to the approach espoused by Lamport[2], who suggests authors of documents should only specify the structure of a document and then allow a typesetting system to present the document structure in a manner most appropriate for the final output device. Lamport quotes two reasons for this system:

1. leaving the final typographical details to the computer allows the document to be much more portable to different output devices, and
2. concentrating on the structure of a document rather than its appearance is of obvious benefit.

These reasons can equally be applied to Web documents and with even more force. The document will be made available across the network to any number of client interfaces. Each of those interfaces may be running with different display mechanisms: for example, viewing the Web via a line mode browser on a traditional terminal and viewing the Web through NCSA Mosaic using more complex display features. Moreover, as a document may use hypertext links to connect to other documents, the structure of documents becomes much more important and care must be taken not to confuse the reader, "losing" them within the Web.

The concept of providing a WYSIWYG editor seems, at first glance, to directly contradict this argument for a logical view of a document. The mere use of a WYSIWYG editor will demote the importance of logical styles as authors actually see how things look on the screen. However, it is important to see how a document will appear as it is written. When editing a document with raw HTML annotations, the content of the document may be harder to determine, as the HTML directives are mixed in with the text of the document. Also, the author should not need to know the exact syntax of this new language when describing the document and a WYSIWYG system allows this syntactic bother to be handled by the computer. It also provides the guarantee that documents are written in fully conformant HTML code. There is constant debate as to the correctness of this approach within various forums, however the authors believe that the logical approach is the correct one with regard to Web documents, although it may not be the easiest model for an author to grasp.

3 Current HTML Editing Systems

Several packages for editing HTML have already been written, based upon two approaches:

1. providing a conversion facility from some typesetting language (e.g. LaTeX or RTF) into HTML, thus allowing authors to use their normal editing facilities.

2. giving a direct WYSIWYG view of the HTML. This method has the benefit of not having to "compile" documents before viewing their final output, and so a document may be viewed incrementally, rather than having to ensure a complete and consistent document at all times: a requirement for any compilation process.

The danger in both of these systems is that there is an inherent loss of information performed by the author as they writes their document if the editor provides only typographical information rather than logical styles. Only heuristics can be used to determine what the author meant when determining the HTML and, by definition, the heuristics will not always be correct.

An important consideration in creating a WYSIWYG editor for the Web is the distinction between editing documents and browsing documents. When browsing a document, hypertext links should be "live" and selecting them will attempt to retrieve a new document specified by the link. However when editing a document, selecting an anchor should not perform any actions other than placing the edit cursor at that point. Ideally, browsers and editors should be combined into a single system, but with a very obvious distinction between the different modes. This would allow users to collect documents via the Web and begin editing them, all in a location transparent manner.

Following on from allowing the editor to be connected to the Web, there arises the problem of creating new documents. When attempting to follow a hyperlink and the response "Not Found" is returned, should it be a browsing error, or a cue to create a new document?

3.1 TkWWW

This system is a fully functional Web browser which supports editing of retrieved documents. The editor is written completely in the Tcl/Tk toolkit and so may be easily enhanced to increase functionality. The editor supports all major HTML functions (character emphasis, headings, paragraph breaks, lists, anchors, etc). Generally, documents can be written as in any editor, and the hypertext links either added in afterwards or placed as you go along. Links and anchors (which are added in the same manner) may be added to any text. However, once a hyperlink is placed in the document, it becomes tricky to edit the text referencing the link, as documents edited in TkWWW are "live" and connected to the Web. The newly placed hyperlink will immediately become active.

Lists are handled in a simple fashion. A "list item" may be inserted at any point making the following text a list item. There is no facility to take a number of lines of text and turn them into a list, nor is there the obvious provision to do lists-in-lists. Two list types are supported: list items (<LI>) and glossary items (<DD>). Selecting a glossary item also generates a term entry (<DT>). Lists are automatically enclosed in either UL or DLs, but if the types are mixed, these inclusions are not nested.

Images can be inserted and marked as anchors. However, textual alternates (if the viewer cannot display images) cannot be specified and nor can attributes such as "ismap" be set.

Overall appearance is good and editing simple. All general functions are available and the ability to browse documents while editing is useful (but it would be better if it did not conflict with moving the cursor for editing).

3.2 Using Word for Windows (CU-HTML)

The Chinese University of Hong Kong has produced a template-based system for Word for Windows which allows the author to use a commonly available editor to produce their documents. Providing a template for a standard word-processor means the author does not have to learn a new editor and can almost immediately begin to create and edit documents. Most of the additional features provided by HTML are implemented as either menu options (eg. add URI) or as styles (eg. Normal, Heading1, etc.).

The editor allows for all the standard HTML formatting, including lists within lists (although the author is forced to specify which list is at which level as the template cannot compute this). It is also possible to include and display inline GIFs in Word. Inclusion of other image formats is not possible. Creating hyperlinks around either text or images is supported. However, such attributes as "alt" and "ismap" for images cannot be added. It also appears impossible to place named anchors into a document.

By default, documents are saved in standard Word format. An option is available to save html documents. The actual HTML is somewhat inaccurate, with head and body directives missing.

4 The Htmltext Editing Package

The system implemented at City University is a full implementation of an HTML editor. It is written using the Andrew Toolkit (ATK), which has excellent in-built facilities for writing a WYSIWYG editor, allowing the programmer to concentrate on other, more important issues.

4.1 The Andrew Toolkit

ATK is a graphical user interface toolkit which provides an object oriented toolkit, written in C, from which objects can easily be combined into complex multi-media applications[3, 4]. Example applications distributed with ATK include a full text editor, a multi-media (MIME) mail interface, a hypertextual help system and an editor for graphical figures. All objects within the ATK system are dynamically loadable, so an application built with ATK can incorporate newly developed objects on the fly.

ATK provides, as a basic building block, a text object which allows full WYSIWYG editing, using well-known emacs or vi bindings. It allows multiple physical styles of text to be freely mixed and allows other ATK objects to be embedded within. Also, the styles which may be placed over text are represented symbolically and the user can customise the appearance of any style by using an editor provided for this purpose, or by simply editing the *template* which is a non-complex textual file. The ATK text object is used as the basis for the HTML editing package described in this paper.

ATK is a stable system which provides many rich features desirable in a WYSIWYG editing package. It has been extensively tested on a variety of platforms and so re-using the objects from its toolkit should provide obvious benefits of robustness.

4.2 The Htmltext Object

The system we have implemented for editing HTML is an ATK object, called *htmltext*, which can represent HTML tags as textual styles within the ATK text object. In theory, it is possible to create a simple template for the standard ATK editor and then with the aid of a converter, possibly bolted onto the editor itself, provide an HTML editing system with far less complexity. We have already presented an argument as to why we believe converting, or compiling, documents is not the best solution for HTML. However, we also believe that an object dedicated to understanding HTML is of more use, especially as HTML grows more complex (see the discussion of HTML+ later) and as the object itself can then be used by other applications, without having to know about HTML conversion. The htmltext object can parse any HTML document to produce a WYSIWYG view with all of the correct logical formatting. While the document is displayed, the user can freely edit any of the text and add or delete any styles from any part of the document. By default, all of the logical styles are configured to look similar to the NCSA Mosaic representation.

A example screen from the editor is shown in figure 1, from which there are a few immediate observations:

- most obviously, both paragraph and line breaks are optionally visible within the text. Paragraph markers are denoted by the normal symbol and line breaks are denoted by the

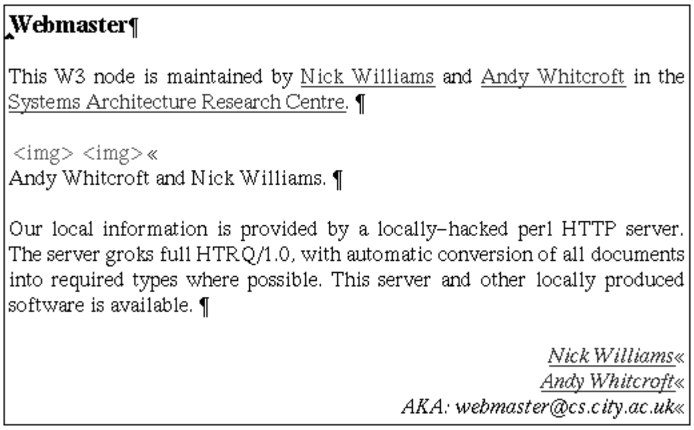


Figure 1: Using the editor on our local webmaster page found at http://web.cs.city.ac.uk/webmaster.html

double left angle bracket symbol. Initially, these were absent in the editor and it used heuristics to determine the location of the paragraph breaks. However this could cause a large discrepancy between the edited text and the final output: the editor itself uses extra newlines to indicate extra paragraph spacing and similar effects. The compromise we chose was to let the author place newlines in arbitrary places, but to only recognise paragraph markers where explicitly stated. Also, as line breaks are now visible in the editor, it forces authors to realise the common error of using many line breaks within a paragraph, not realising that the editor or browser will perform line breaks for them.

- hypertext anchors are not distinct buttons within the text but are indicated by simple physical styles placed over the text referencing the link. This allows the text to be edited in a completely normal fashion, allowing it to flow more easily with the surrounding text. If the anchor were represented by a distinct button then the text of the link would become separated from the body, allowing authors to create documents with more stilted text.
- inline images are not currently displayed within the document as true images. Instead markers are placed at their position in the text.

4.3 Features of the htmltext object

By creating the htmltext object as a subclass of the standard text object in ATK, many features are automatically inherited.

- The document can be printed without having to create any special case code defining how to print the object.
- A table of contents can be automatically generated, showing all headers and hence the structure of the document. This is provided by a special object which can examine any text object and determine the structure based on an ordered list of "heading" styles. As the htmltext object is derived from the text object, this table of contents operation can be performed on the htmltext.

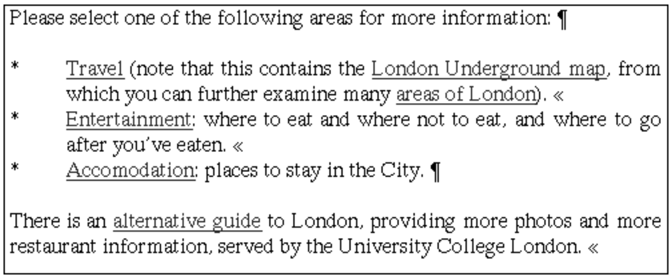
- Spell checking is provided by the editor object in a similar manner to the table of contents generator. As the htmltext contains standard text data, the spell checker can process a document as if it were a totally normal textual file, while ignoring all of the HTML directives.
- The display of the different styles is entirely customisable on a per-user basis.
- More generally, any function which has been, or will be written to operate on the standard text object will be able to work with the htmltext object.

4.4 Lists

One of the most complex features of HTML to implement in user-interface terms is creating a list. The HTML annotation describing a list is at two levels:

1. a surrounding tag indicating the type of list. There are several types of list: itemised, ordered/enumerated, glossaries/descriptions, menus and directories.
2. every item within the list must also be tagged to show its start. This item tag is not the same for all lists. For glossary lists, this is further complicated by having to distinguish the term which is being defined and the definition itself. The HTML for describing these lists may appear simple, but is in fact difficult to specify within a user interface, while still providing an easy-to-use system for the authors.

The editor supports all of the various list types. The method of annotating a section of text to be a list is to select a region and then to request a specific list to be placed over that region. Each line in the selection will be made into an appropriate list item, and the correct list environment wrapped around the whole. When no selection is present and a list environment is requested, the current line is made into a list. If the current line is located within or immediately adjacent to a list environment of the same type, it is merged with that list. This allows you to incrementally add items to lists, and by using the selection method it allows sub-lists to be created.


Please select one of the following areas for more information: ¶

* Travel (note that this contains the London Underground map, from which you can further examine many areas of London). «
* Entertainment: where to eat and where not to eat, and where to go after you've eaten. «
* Accomodation: places to stay in the City. ¶

There is an alternative guide to London, providing more photos and more restaurant information, served by the University College London. «

Figure 2: An extract showing the appearance of a bulleted list within the htmltext object. The document being viewed is the toplevel menu from CityWeb's London Guide (http://web.cs.city.ac.uk/london/guide.html).

Description, or glossary, lists are handled slightly differently. The system for creating lists is identical, although each list item is marked as being a glossary definition and the glossary term element (<DT>) is ignored at this point. The author can place the glossary term at any point in the text at any time.

Cutting and pasting of text between different HTML applications happens for free. This can also be a bug since, as yet, there is no provision for pasting plain text into the editor. Any text taken from the cut buffer is assumed to be HTML formatting, complete with paragraph breaks. If the text is not formatted and hence has no paragraph markers, it will be combined into a single large paragraph.

4.5 Implementation Details

4.5.1 HTML Attributes

As has been mentioned briefly in previous sections, there are many attributes defined in HTML which may be attached to a normal tag to change its behaviour. For example, there is an attribute "compact" which, when placed on a list, is supposed to cut down on the spacing used. Attributes are advisory only, merely indicating a preference for how a tag should be displayed within the client. Many editors ignore the attributes, or special-case them. For example "href", the attribute which specifies the destination of a hyperlink is implemented in almost every HTML editing system. However, other attributes, such as "compact", are completely ignored and cannot be placed onto a tag, even though there is no conceptual difference between the two attributes. During the implementation of the htmltext object, this became evident and a mechanism for placing generic attributes was sought and implemented, using a feature provided by the standard objects within ATK.

The ATK text object keeps physical style information and the textual data separately, so the text stream can be manipulated easily, ignoring how text should be formatted. The physical styles are in a separate data structure, defining the areas for which the style is active, the name of the style and any attributes the application may wish to attach to a style. The style name is kept symbolic throughout the system, so that any changes to the appearance of that style may take effect immediately. Styles may have arbitrary attributes attached to them, where attributes are tuples of strings. This feature maps extremely well to the HTML model of attributes and the htmltext object uses the feature for this. A dialog box system is provided to place or edit the attributes in the document. This system does not require

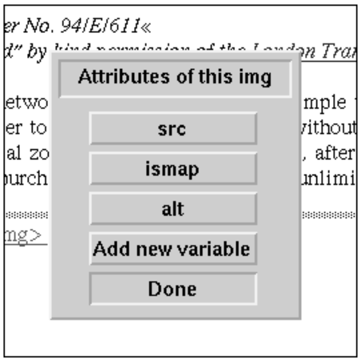


Figure 3: A screenshot showing the attribute editor

the editor to have any understanding of what the attributes mean. For example, although images may be placed inline, there is currently no understanding of how they work within the editor. This does not prevent the author from selecting the image and placing "ismap", or "alt" attributes onto it.

4.5.2 Web Browsing Capability

Within this version of the system, only local documents can be edited; retrieval of documents using HTTP is not supported. This also prevents support for inline images: until the data can be retrieved via HTTP, it is fairly useless to provide the facility for displaying an image, even if it were possible. Up to this point, we have not provided HTTP support, as we believe it is a completely separate issue to that of providing an object capable of editing the HTML language. When the HTTP protocol is added into the ATK system, in whatever manner, then the htmltext object will immediately be able to make use of this. It is worth stressing that the htmltext object is not a complete application. It is merely an object which, even in its rudimentary state, can be used to edit HTML with complete functionality. Because of its object nature, it can be placed inside other applications, making them HTML aware: for example when a document is received via e-mail which is recognised as HTML, the ATK mail reader can detect this and automatically use the htmltext to view the document.

4.5.3 Using The Standard Library

The WWW people at CERN distribute a standard library which provides support for retrieving documents via HTTP and for providing HTML parsing within an application. This library has not been used in the implementation of the htmltext object so far. The HTTP retrieval mechanism would be desirable and at some point may well be placed into an application which uses the htmltext object. The HTML parsing provided by the library was looked at during the implementation but the decision was made not to use it. The major reason for this was that the library provides parsing in one direction only. Indeed, it is not easy to envisage how the library would provide functions for creating the HTML, as this is greatly dependent on how the application handles the document internally. However, with this in mind, the application will have to understand HTML at least in the direction of writing and so using a separate library for reading the HTML seems to create more problems than it is worth.

5 Comments Based On The Htmltext Experience

From writing this implementation and seeing the other systems around, it is evident that HTML is not a difficult language to generate, assuming the logical styling information is at least partially available. However, there are many small details of the system which make either generating the language, or providing a simple interface with which to specify logical information, awkward to implement. For example, attributes are used inconsistently. In almost all cases, attributes are simple modifications of a basic tag to make its appearance slightly different; the modifications may be ignored by browsers. However the "href" attribute used to indicate hyperlinks is an extreme exception: it changes the entire behaviour of the text it surrounds. In a similar fashion, the "ismap" attribute of an inline image is a clumsy implementation: the ismap attribute should really be placed onto the hyperlink anchor itself. This kind of inconsistency merely results in implementations of HTML editors which use special cases to ease the job of creating the different tags, without catering to the underlying model.

Also annoying is the manner in which representing a tagged section of text is different for some environments. Most tagged text has very strict begin/end pairs of tags annotating the specific text, but list items use a completely different approach: that of marking the start of the section and not using an end tag. And some items (images and paragraph breaks) are not environments over text at all, but are singleton tags indicating "magic". None of these is difficult to overcome, however collectively they increase the complexity involved in making an engine to both read and write correct code, knowing the difference between all the different types.

It is most desirable that an editor be "attached" to the Web and have the ability to read and write its documents using HTTP. This necessitates servers correctly implementing the CHECKOUT and CHECKIN methods, as these seem the most appropriate for editing documents. Unfortunately few, if any, servers yet implement this feature. The authors know of only one: deceit[5]. Further, it is vital that there should be a distinction between editing and browsing the Web. When a user wishes to edit a document they have retrieved through the web, the browser should CHECKOUT the document from the server before allowing any editing. While editing, the document should not be live, allowing the user to freely select any portion of text and edit it, without worrying about whether the application will attempt to follow any hyperlinks. A multiple buffer browser/editor is preferable, as this would allow the user to continue browsing the Web in another buffer, allowing the possibility of creating hyperlinks using the "point-and-click" interface, instead of using URIs. These ideas are intended to be added into the htmltext editor in the future.

The editor has been in use within City University for a number of months now, with almost all of the CityWeb pages written with its aid. The feedback from users has been very good, despite the many bugs and "features" which have plagued the editor during its development.

6 Implementing HTML+ with ATK

The new emerging language to "replace" HTML is the superset language HTML+[6]. This language has very similar structure to its predecessor, but with a few notable differences.

- Firstly, paragraphs within HTML are indicated by markers denoting the end of a paragraph. This means that determining the exact extent of a paragraph involves remembering where the previous paragraph ended (or a paragraph was implied by something else!). In HTML+, paragraphs are tagged sections of text, marking the exact beginning and end. This also allows attributes to be placed onto the paragraph tag, such as directives to place the paragraph into a footnote or margin note, or to mark the paragraph as having particular relevance such as a warning note. In theory, these attributes could be placed onto the HTML paragraph break tag, however because of the inability to work out the exact extent of the paragraph, this would be awkward to implement.

- Rather than have multiple different logical styles for types of emphasised text, HTML+ combines all of the logical and physical styles into one: <EM>, which should have an attribute specifying the desired physical style of emphasis.

- HTML+ contains a number of new features such as formatted multi-column tables and more complex figures and diagrams.

ATK has a fairly well established spreadsheet type object, which allows arbitrary tables to be created. Like most ATK objects, the spreadsheet object (known as *table*) may be embedded within a text object, and may itself have other objects (such as text) embedded in cells within the spreadsheet. This could map easily to the HTML+ table environment.

A fairly recent addition to the ATK library of objects is an image object capable of displaying colour images from JPEG or GIF data. By embedding this within the htmltext object, inline images could easily be shown. More complex images (and overlaying images) could be represented by the structured drawing object. This has already been implemented in the htmltext object but it will not be useful until HTTP is integrated into the system.

All of these ATK objects are well-established items which can easily be integrated within the HTML editing package. More objects exist which have not been mentioned: a complex equation editor could be provided, or a programming language object could be embedded, providing features similar to those found in the new Viola interface.

7 Conclusions

An editing package for HTML has been described which allows WYSIWYG editing of HTML documents. By using the Andrew User Interface System to develop the package, the potential of the editing package is large, allowing it to be extended to HTML+ and to be integrated into various applications such as a Web browser, a Web editor or even completely different applications such as a mail reader or a help system. Some problems have been pointed out with the difficulties in combining Web editors and browsers, although the authors are currently developing a further implementation of the htmltext package which will successfully merge the two functions.

Feedback from users of the editor has so far been good, although more work needs to be done in the area of the user interface of such an editor, where the dichotomy of semantic markup and physical styles present a difficult combination. Many other systems for producing HTML, either by a special editor or by providing conversion tools, have been implemented. However few real WYSIWYG, direct access systems have been produced so far, probably because of the aforementioned difficulties of handling browsing and of providing a reasonable interface onto logical markup.

The htmltext object is available both in source form and packaged up into a precompiled binary for sun4 systems. The ftp site is `ftp.cs.city.ac.uk` and the distributions are to be found in the directory `/pub/htmltext`.

References

[1] IETF, *Hypertext Markup Language (HTML)*, Working Draft.

[2] Lamport, Leslie. *LaTeX: A Document Preparation System*, Addison-Wesley, 1986.

[3] Borenstein, Nathaniel S., *Multimedia Applications Development with the Andrew Toolkit*, Prentice Hall, 1990.

[4] Palay, A., W. Hansen, M. Kazar, M. Sherman, M. Wadlow, T. Neueundorffer, Z. Stern, M. Bader, T. Peters, *The Andrew Toolkit: an Overview*, Proceedings of the USENIX Technical Conference, Dallas, February, 1988.

[5] Whitcroft, A. and Wilkinson, T. *A Tangled Web of Deceit*, Proceedings of WWW'94, Switzerland.

[6] Dave Ragget, Hewlett Packard, *HTML+ Discussion Document.*