HTML macros - Easing the construction and maintenance of Web texts [1]

Andrew Peel (A.T.Peel@ukc.ac.uk)
Computing Laboratory
University of Kent at Canterbury
Canterbury, Kent CT2 7NF
United Kingdom

Telephone: +44 1227 82 3979 Fax: +44 1227 762 811

18th December 1995

Abstract

Authoring and maintaining large collections of Web texts is a cumbersome, error-prone and time-consuming business. Ongoing development of courseware for the High Performance Computing Consortium (HPCC) TLTP [2] has only helped to emphasise these problems.

Courseware requires the application of a coherent document layout (templates) for each page, and also the use of standard icons with a consistent functionality, in order to create a constant look and feel throughout the material. This provides the user with an environment where he or she can access new pages, and instantly recognise the format used, making the extraction of the information on the page much quicker, and less immediately confusing.

This paper describes a system that was developed at UKC to provide a solution to the above problems via the introduction of HTML macros. These macros can be used to provide a standard document layout with a consistent look and feel, as well as tools to ease user navigation.

The software is written in Perl, and achieves macro expansion and replacement using the Common Gateway Interface (CGI) and filtering the HTML source.

Using macros in your HTML results in your document source code being shorter, more robust, and more powerful. Webs of documents can be built extremely fast and maintenance is made much simpler.

Keywords: Authoring, Automation Tools, Perl filters for HTML, Teaching and learning on the Web

[1] A Hypermedia Seminar based on this paper was given at UKC on the 12th July 1995

[2] TLTP is funded by the HE funding councils of England, Scotland, Wales and Northern Ireland

1 Introduction - Building a web text

By web text, we mean:

web text = content + hyper-structure

That is to say a web text consists of its text, structured with formatting tags, and its hyperlinks to itself and other, external web texts.

There are currently three ways to build such a web text:

1. By hand, using a text editor such as *vi* or *MS Notepad* to both write the content and to insert HTML markup tags. This gives the author complete control over construction at the lowest level feasible.
2. HTML assistants, such as *HoTMetaL*, or *MS Word's Internet Assistant* provide extra functionality to help increase productivity. They can be used to validate HTML as it is entered, as well as check and maintain hyperlinks. They can also provide other typical editing application benefits, such as spell checking, and a WYSIWYG display (or at least WYSIWYAG (What You See Is What You Almost Get)).
3. HTML convertors, such as *LaTeX2HTML* [Dra94], possibly the best example of its kind. It takes an existing LaTeX document and not only converts the formatting markup into the equivalent HTML markup, but extracts its structure and uses it to build a hierarchical web. It also adds facilities to navigate this new web. These aspects of the package can be seen in figures 1 and 2. The former is the "root" node of the original LaTeX document, and the latter is the result of following the "Structure Extraction by Tree Search" hyperlink from the root node.

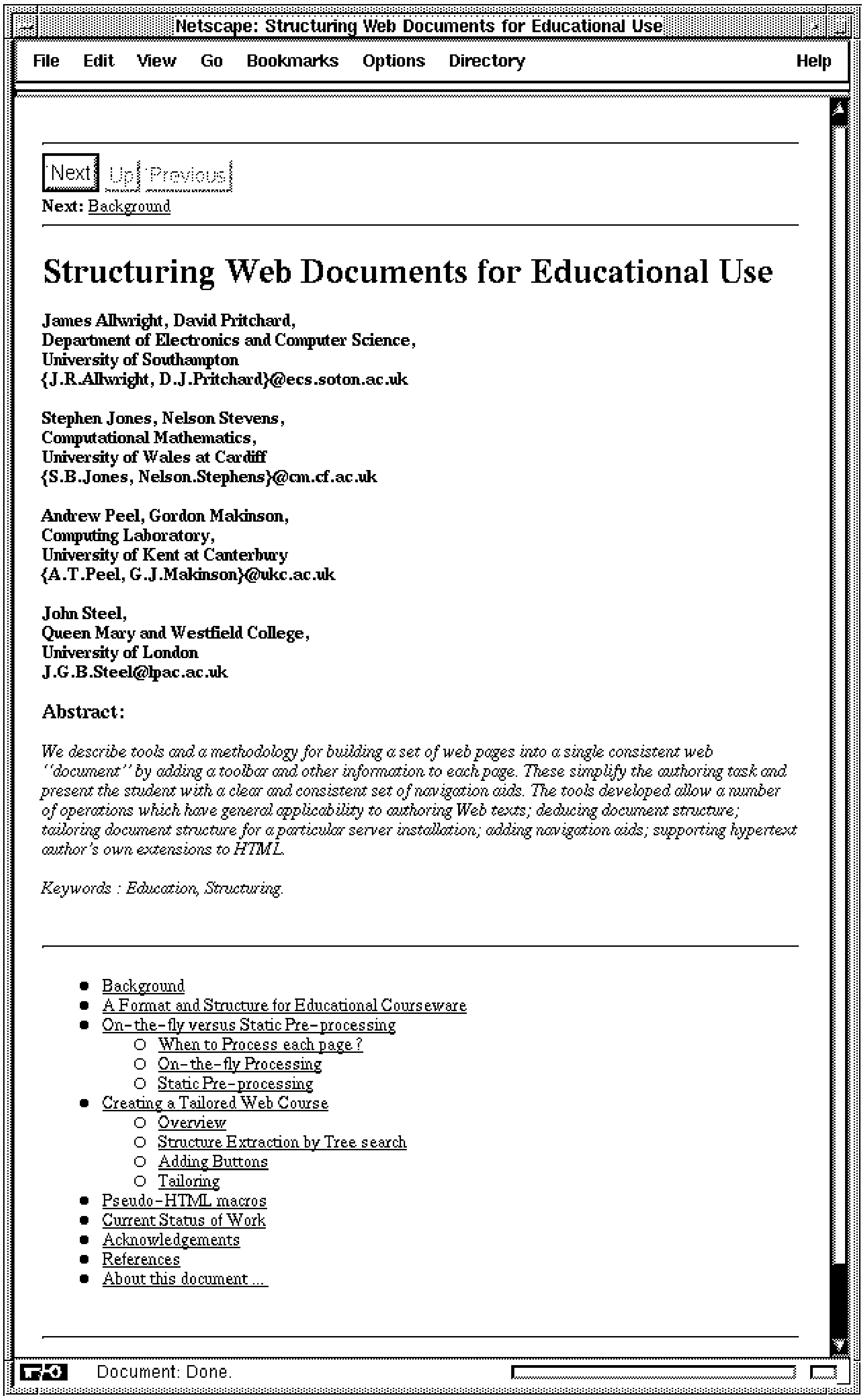
HTML assistants are currently embryonic [Web95] but will surely evolve into highly useful tools as they start to include more features, such as drag and drop WYSIWYG editing, which will soon be seen with the 2.0 Gold release of the Netscape Navigator [Net95].

LaTeX2HTML is undoubtedly useful, as it can turn LaTeX documents into web texts, providing structure, hyperlinks and a navigation button bar. It even converts LaTeX mathematical equations into GIF format so they are not lost from the web text until HTML Level 3 browsers become stable and more common. Most importantly, it is an automated process, with no need for the user to have any knowledge of the underlying conversion routines, or even of any aspect of web publishing.

The user is forced into using a tree hierarchy, which may be inappropriate and inflexible, but it does reduce the probability of users getting lost in the web. As Brown says, "A hierarchy is a powerful aid to orienting the user – indeed it is the only higher-level abstraction he has" [Bro91]. Berners-Lee et al. go further, saying "for the information provider such systems are easy to build by cross-linking existing filesystems" [BLCGP92]. That is to say that hierarchical web texts can mirror hierarchical file systems.

Once converted into HTML, the web nodes don't lend themselves easily to manual maintenance or extension. This is due in part to the sheer volume of HTML added to create the navigation button bar, etc. and in part to the automation of filename creation. These filenames have no semantic meaning, and are hence difficult to remember.

Tailoring of the nodes can be achieved *during* conversion from their LaTeX source by using *LaTeX2HTML* command line arguments and template files.


Netscape: Structuring Web Documents for Educational Use

File Edit View Go Bookmarks Options Directory Help

Next Up Previous

Next: Background

Structuring Web Documents for Educational Use

James Allwright, David Pritchard,
Department of Electronics and Computer Science,
University of Southampton
{J.R.Allwright, D.J.Pritchard}@ecs.soton.ac.uk

Stephen Jones, Nelson Stevens,
Computational Mathematics,
University of Wales at Cardiff
{S.B.Jones, Nelson.Stephens}@cm.cf.ac.uk

Andrew Peel, Gordon Makinson,
Computing Laboratory,
University of Kent at Canterbury
{A.T.Peel, G.J.Makinson}@ukc.ac.uk

John Steel,
Queen Mary and Westfield College,
University of London
J.G.B.Steel@lpac.ac.uk

Abstract:

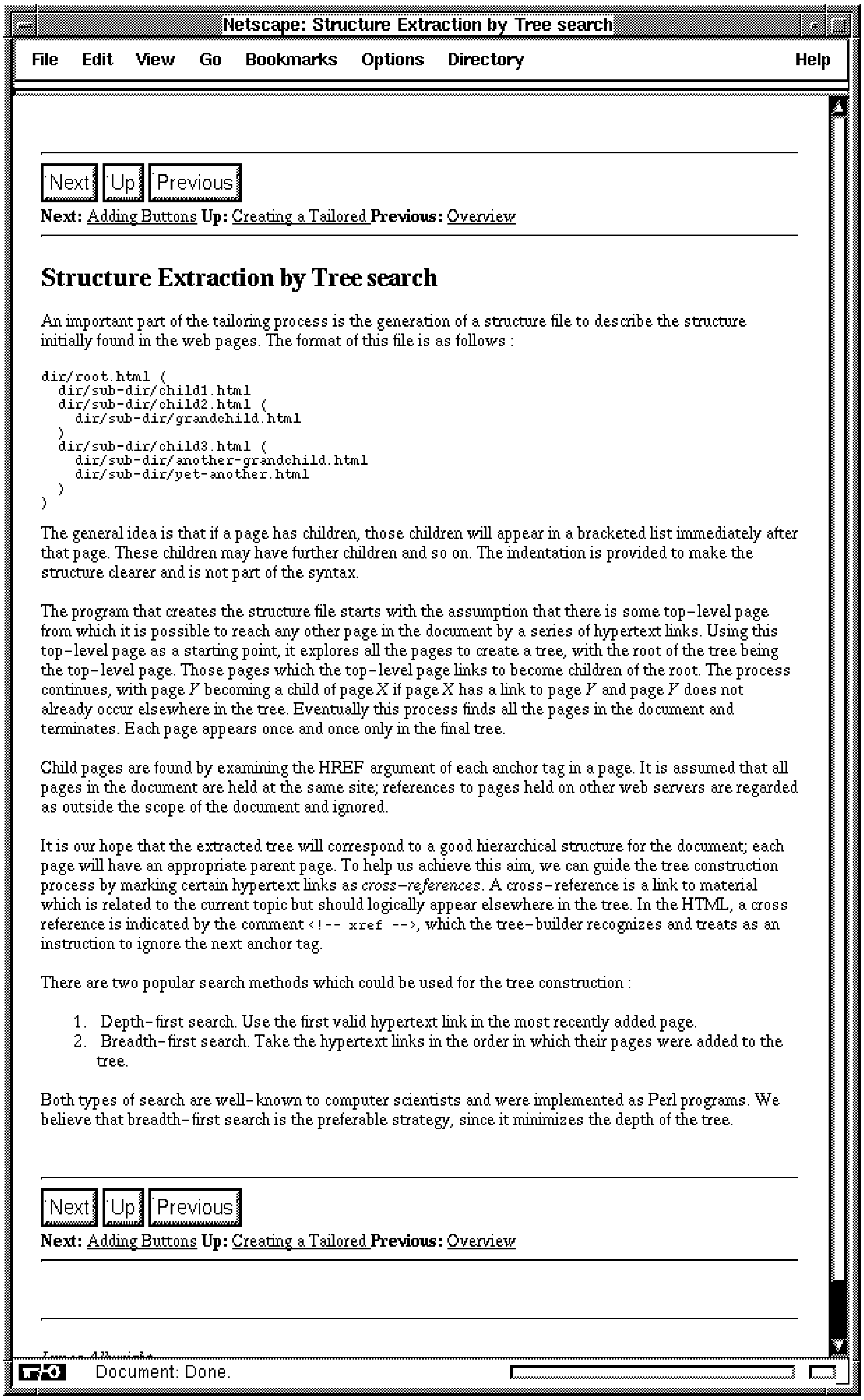
We describe tools and a methodology for building a set of web pages into a single consistent web ''document'' by adding a toolbar and other information to each page. These simplify the authoring task and present the student with a clear and consistent set of navigation aids. The tools developed allow a number of operations which have general applicability to authoring Web texts; deducing document structure; tailoring document structure for a particular server installation; adding navigation aids; supporting hypertext author's own extensions to HTML.

Keywords : Education, Structuring.

- Background
- A Format and Structure for Educational Courseware
- On-the-fly versus Static Pre-processing
  - When to Process each page ?
  - On-the-fly Processing
  - Static Pre-processing
- Creating a Tailored Web Course
  - Overview
  - Structure Extraction by Tree search
  - Adding Buttons
  - Tailoring
- Pseudo-HTML macros
- Current Status of Work
- Acknowledgements
- References
- About this document ...

Document: Done.

Figure 1: *LaTeX2HTML* root node



Netscape: Structure Extraction by Tree search

File Edit View Go Bookmarks Options Directory Help

Next Up Previous

Next: Adding Buttons **Up:** Creating a Tailored **Previous:** Overview

Structure Extraction by Tree search

An important part of the tailoring process is the generation of a structure file to describe the structure initially found in the web pages. The format of this file is as follows :

```
dir/root.html (
  dir/sub-dir/child1.html
  dir/sub-dir/child2.html (
    dir/sub-dir/grandchild.html
  )
  dir/sub-dir/child3.html (
    dir/sub-dir/another-grandchild.html
    dir/sub-dir/yet-another.html
  )
)
```

The general idea is that if a page has children, those children will appear in a bracketed list immediately after that page. These children may have further children and so on. The indentation is provided to make the structure clearer and is not part of the syntax.

The program that creates the structure file starts with the assumption that there is some top-level page from which it is possible to reach any other page in the document by a series of hypertext links. Using this top-level page as a starting point, it explores all the pages to create a tree, with the root of the tree being the top-level page. Those pages which the top-level page links to become children of the root. The process continues, with page Y becoming a child of page X if page X has a link to page Y and page Y does not already occur elsewhere in the tree. Eventually this process finds all the pages in the document and terminates. Each page appears once and once only in the final tree.

Child pages are found by examining the HREF argument of each anchor tag in a page. It is assumed that all pages in the document are held at the same site; references to pages held on other web servers are regarded as outside the scope of the document and ignored.

It is our hope that the extracted tree will correspond to a good hierarchical structure for the document; each page will have an appropriate parent page. To help us achieve this aim, we can guide the tree construction process by marking certain hypertext links as *cross-references*. A cross-reference is a link to material which is related to the current topic but should logically appear elsewhere in the tree. In the HTML, a cross reference is indicated by the comment `<!-- xref -->`, which the tree-builder recognizes and treats as an instruction to ignore the next anchor tag.

There are two popular search methods which could be used for the tree construction :

1. Depth-first search. Use the first valid hypertext link in the most recently added page.
2. Breadth-first search. Take the hypertext links in the order in which their pages were added to the tree.

Both types of search are well-known to computer scientists and were implemented as Perl programs. We believe that breadth-first search is the preferable strategy, since it minimizes the depth of the tree.

Next Up Previous

Next: Adding Buttons **Up:** Creating a Tailored **Previous:** Overview

Document: Done.

Figure 2: *LaTeX2HTML* child node

2 Requirements for a web text construction tool

Now we've looked at the tools currently available, we can make some generalisations about expectations of such a construction tool.

- To make life easier for the *end-user* by providing a navigation facility and by producing a standard look and feel using templates.
- To make life easier for the *developer* by making web texts easier and quicker to produce, and more robust and easier to maintain.
- To do all this without having to write a new web browser (or server)!

To paraphrase: what is needed is to provide a consistent and robust service to end-users of HTML (i.e. web browser users) whilst reducing the amount of work the developer has to do, without reinventing the wheel in the process.

3 Solution: HTML Macros

Nobody in their right mind writes in raw troff or TEX. Instead, a higher level of control is attained through the use of troff ms (or mm) macros, or LATEX, respectively.

In the same way, HTML is crude and cumbersome. Macros aim to reduce the amount of manual markup, making web texts more robust, easier to maintain, and quicker to build. They do this by using standard HTML tags as building blocks for more more complex and sophisticated components. Uses include the following:

- Building templates
- Creating HTML on-the-fly from plain text files or databases
- Helping novice users create web material with little or no knowledge of HTML, SGML, DTDs, etc.
- Reuse of common HTML fragments

Hence macros can be used to create a higher level of interaction with HTML for the web text developer.

For example, macros can be used to:

- build common sequences of HTML tags
- store common hyperlinks (to save incorrect reentry of URLs)
- define data templates for frequently used CGI tools (see section 7, Case study: The glossary button (Tiler as a bolt-on filter))

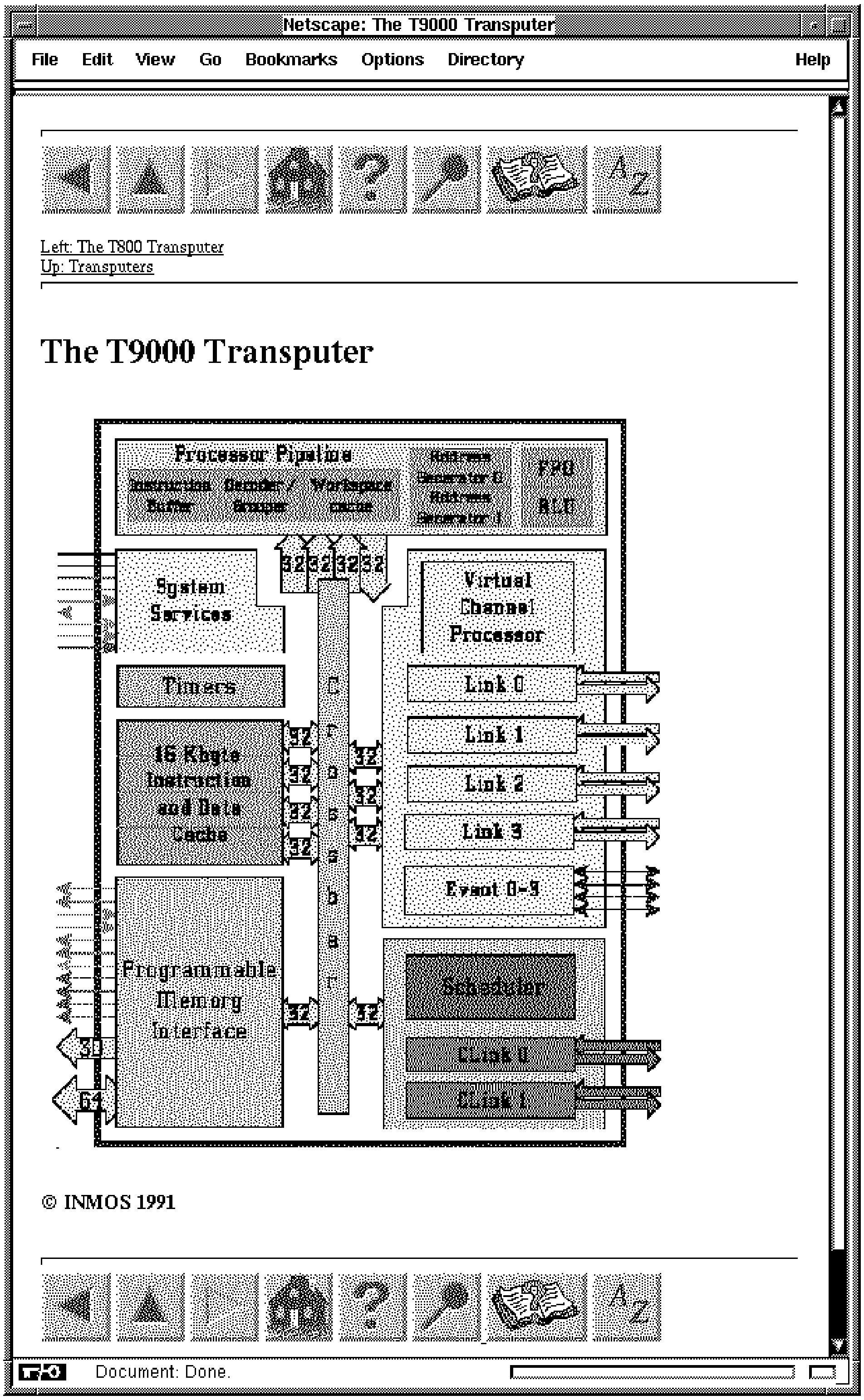


Figure 3: The T9000 Transputer web node

4 Case study: The T9000 Transputer

To create the T9000 Transputer web node of figure 3 requires the following HTML (it's barely readable, but you can see the idea):

```
<HR SIZE=5>
<A HREF="/courseware/sections/transputers/T800.html"><IMG SRC="/sicons/Prev.gif" BORDER=0 ALT="Previous page"></A>
<A HREF="/courseware/sections/transputers/"><IMG SRC="/sicons/Up.gif" BORDER=0 ALT="Up page"></A>
<IMG SRC="/sicons/Next_dead.gif" BORDER=0 ALT="No next page">
<A HREF="/courseware/"><IMG SRC="/sicons/icon.home.gif" ALIGN=TOP BORDER=0 ALT="Home"></A>
<A HREF="/courseware/sections/help/&barpos=defaults"><IMG SRC="/sicons/icon.help.gif" BORDER=0 ALT="Help!"></A>
<A HREF="/cgi-bin/tiler/search"><IMG SRC="/sicons/icon.search.gif" BORDER=0 ALT="Search"></A>
<A HREF="/courseware/sections/library/"><IMG SRC="/sicons/icon.lib.gif" BORDER=0 ALT="Library"></A>
<A HREF="/courseware/sections/info/"><IMG SRC="/sicons/icon.index.gif" BORDER=0 ALT="Index"></A>
<BR><BR>
<A HREF="/courseware/sections/transputers/T800.html">Left: The T800 Transputer</A><BR>
<A HREF="/courseware/sections/transputers/">Up: Transputers</A><BR>

<HR SIZE=5>
<P><BR>

<H1>The T9000 Transputer</H1>
<P><BR>



<IMG SRC="/courseware/sections/transputers/images/T9000.gif"  ALT="Image of the T9000 Transputer">
<P><BR>
<H3>&copy INMOS 1991</H3>

<P><BR>
<HR SIZE=5>
<A HREF="/courseware/sections/transputers/T800.html"><IMG SRC="/sicons/Prev.gif" BORDER=0 ALT="Previous page"></A>
<A HREF="/courseware/sections/transputers/"><IMG SRC="/sicons/Up.gif" BORDER=0 ALT="Up page"></A>
<IMG SRC="/sicons/Next_dead.gif" BORDER=0 ALT="No next page">
<A HREF="/courseware/"><IMG SRC="/sicons/icon.home.gif" ALIGN=TOP BORDER=0 ALT="Home"></A>
<A HREF="/courseware/sections/help/"><IMG SRC="/sicons/icon.help.gif" BORDER=0 ALT="Help!"></A>
<A HREF="/cgi-bin/tiler/search"><IMG SRC="/sicons/icon.search.gif" BORDER=0 ALT="Search">
</A><A HREF="/courseware/sections/library/"><IMG SRC="/sicons/icon.lib.gif" BORDER=0 ALT="Library"></A>
<A HREF="/courseware/sections/info/"><IMG SRC="/sicons/icon.index.gif" BORDER=0 ALT="Index"></A>
<BR><BR>
<A HREF="/courseware/sections/transputers/T800.html">Left: The T800 Transputer</A><BR>
<A HREF="/courseware/sections/transputers/">Up: Transputers</A><BR>

<HR SIZE=5>
```

This creates both the button bars, links to parent and sibling nodes and includes the image of the T9000 Transputer.

To save typing, the HTML of the button bar could be saved as a separate file, and loaded into each web node as necessary. The tailoring that would have to follow to update the links, buttons available, etc. in each node could easily lead to errors creeping in, and would certainly take a considerable amount of time.

The following HTML contains a macro to create the button bar. It is functionally equivalent to the above section of HTML, but is significantly shorter.

```
<BAR TITLE="The T9000 Transputer" LEFT="T800.html" UP="/courseware
/sections/transputers/" ALIGN="both">

<IMG SRC="images/T9000.gif" ALT="Image of the T9000 Transputer">
<P><BR>
<H3>&copy INMOS 1991</H3>
```

As you can see, it's much more concise.

This `BAR` macro creates a web node with a title of "*The T9000 Transputer*", a parent node with a filename of "*/courseware/sections/transputers/*", a single immediate sibling at "*T800.html*" and button bars at both the top and bottom of the node.

The full syntax of the `BAR` macro is described in table 1.

To run through the various attributes briefly: `TITLE` is used to make an <H1> HTML

Attribute Name(s)	Attribute Value Type
TITLE	string
LEFT, UP, RIGHT	URL
ALIGN	top | bottom | both
LEFTTITLE, UPTITLE, RIGHTTITLE	string
NONAV, NOLIB, NOHELP, NOHOME, NOSEARCH, NOINFO	-

Table 1: The full syntax of the `BAR` macro

header and `<TITLE>`; `LEFT`, `UP` and `RIGHT` are used to specify parent and sibling URLs; `ALIGN` specifies the position of the button bar(s); `LEFTTITLE`, `UPTITLE` and `RIGHTTITLE` are used to explicitly specify titles (the title look-ahead routine cannot look into the dynamic entrails of CGI scripts for titles); the `NO-` attribute values turn off their respective buttons on the button bar.

To understand how the macros are expanded, we need to look at how the Common Gateway Interface works.

5 How does the CGI work?

The Common Gateway Interface (CGI) [McC93] facilitates the provision of interactive web texts through HTML forms, and web texts that can be created on-the-fly.

It does this in the following way:

When a browser requests a file from the server the server loads the file from its filestore (assuming the file exists) and returns two pieces of information to the client. The first information chunk contains various data fields, including the MIME type (e.g. text/html, text/plain, video/mpeg). This informs the client of the type of the requested file, which is then returned as the second information chunk.

For example, if a file was requested that has a video file type, the browser recognises the file type and starts up an appropriate application to handle the received object.

However, a request made through the CGI works slightly differently (see figure 4). When a server receives a request for a file held in a particular directory (normally called *cgi-bin*) it loads the file and executes it. It is the output from this execution that is returned to the client.

Hence files in the CGI bin are executable pieces of code, rather than HTML or data files. They can be written in any available language, such as C, Perl, Tcl, etc. and can therefore provide dynamic web pages, including, for example, the current date and time when the page was generated by the CGI script.

(Note that a CGI script must explicitly output a MIME type, rather than relying on the server to do so.)

6 How does Tiler work?

The Tiler macro processor is written in Perl (Practical Extraction and Reporting Language) [WS92].

HTML files containing Tiler macros are held in a web server's file store. They are not requested directly, as with regular HTML files, but through the CGI.

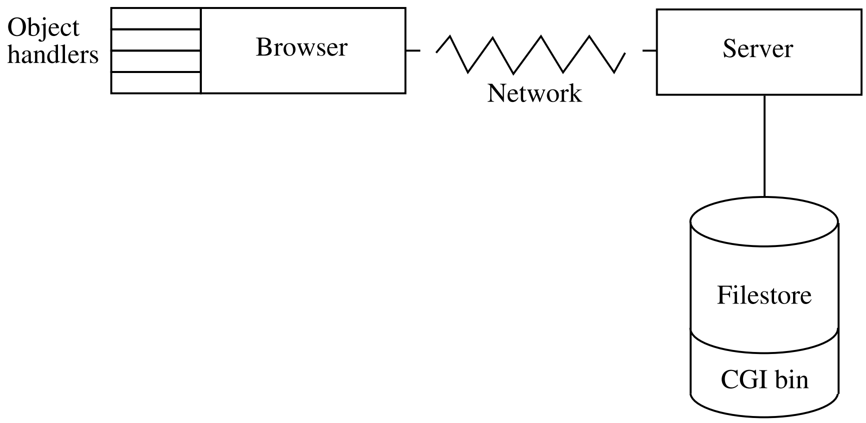


Figure 4: The Common Gateway Interface

For example, the T9000 web node above could be accessed with the URL:

```
http://blah.co.uk/cgi-bin/tiler/tiler?
file=/courseware/sections/transputers/T9000.html
&barpos=defaults&exitpos=both&exitstat=off
```

The web browser passes four parameters to the server via the CGI GET method [McC93] (i.e. information is passed by appending it to the URL, preceded by a question mark).

The above URL results in a script called *tiler* in the server's *cgi-bin/tiler* directory being executed. This script reads in the parameters (separated by ampersands) that follow the question mark in the above URL. These are:

- the name of the file to be filtered
- the position of the button bar (can be used to override the `BAR ALIGN` attribute)
- the position of the emergency exit button
- the status of the emergency exit button

For the moment we will concern ourselves with just the first parameter. Tiler loads the specified HTML file and expands any macros it has been programmed [3] to recognise. It then links any URLs back to the Tiler macro processor.

That is to say that a URL found in an HTML source file, such as:

```
/courseware/sections/transputers/index.html
```

must be changed to:

```
http://blah.co.uk/cgi-bin/tiler/tiler?
file=/courseware/sections/transputers/index.html
&barpos=defaults&exitpos=both&exitstat=off
```

so that when the hyperlink is activated, the file is passed through the Tiler macro expansion filter before being sent back to the client. In this way, every new page is filtered for macros.

This process of linking back to Tiler is the bulk of Tiler's workload.

[3] Macros are hardwired into the Tiler code

7 Case study: The glossary button (Tiler as a bolt-on filter)

As we have seen, Tiler receives the name of the file to be filtered via the CGI GET method (as part of a URL).

Tiler can also accept files on its standard input. This allows other CGI tools to create HTML containing macros on-the-fly, and pass them to Tiler. For example, the Tiler glossary tool.

A glossary button can be created using a Tiler `GLOSS` macro of the following format:

```
<GLOSS WORD="string" KEY="string">
```

For example:

```
<GLOSS WORD="Arithmetic Logic Unit (ALU)" KEY="alu">
```

When included in an HTML file and filtered by Tiler, the following HTML is delivered to the browser:

```
<A HREF="/cgi-bin/tiler/glossary?key=alu>Arithmetic Logic Unit (ALU)</A>
```

Thus the browser creates a hyperlink to the specified URL; that of another CGI script called `glossary`. The `WORD` parameter is used for the text of the link, and the `KEY` parameter is passed to the glossary script, which uses it to search for the matching key, and hence glossary entry, in a glossary database. This hyperlink can be seen in figure 5.

The matching glossary entry can itself contain HTML and Tiler macros. It is passed from the glossary script's standard output to the standard input of Tiler. Tiler then expands the macros and returns the resulting HTML to the client. This creates the web node of figure 6.

The glossary CGI tool is also capable of producing a list of hyperlinks to every entry in the glossary database (see figure 7). This is done via an empty `KEY` attribute value inside the `GLOSS` macro. If the glossary script doesn't receive a `KEY` parameter via the GET method, it knows to produce the hyper-index instead.

8 Other Tiler functionality

8.1 More macros

Other macros written so far include `BIB`, a bibliography macro, `TRANSWEB`, a macro interface to a web occam compiler [MP95], `BULLET`, for creating coloured bullet points, and others.

8.2 The emergency exit button

The linking of URLs back to the Tiler macro filter as discussed above does not mention cases where URLs found in the HTML/macro source point to material at external sites (i.e. not on the local server where Tiler lives). In this case, Tiler cannot load the file itself, as it's not on the local file store, so it uses *libwww* [lib94], a Perl World Wide Web library, to download the specified HTML file across the web, before filtering it.

Here, Tiler acts not as a macro filter, but as an "*exit*" filter that inserts one or more emergency exit buttons [4]. These provide the user with a hyperlink back to the last page they visited

[4] This facility can be turned off or the position of the exit button changed by altering the `exitpos` and `exitstat` parameters mentioned in section 6

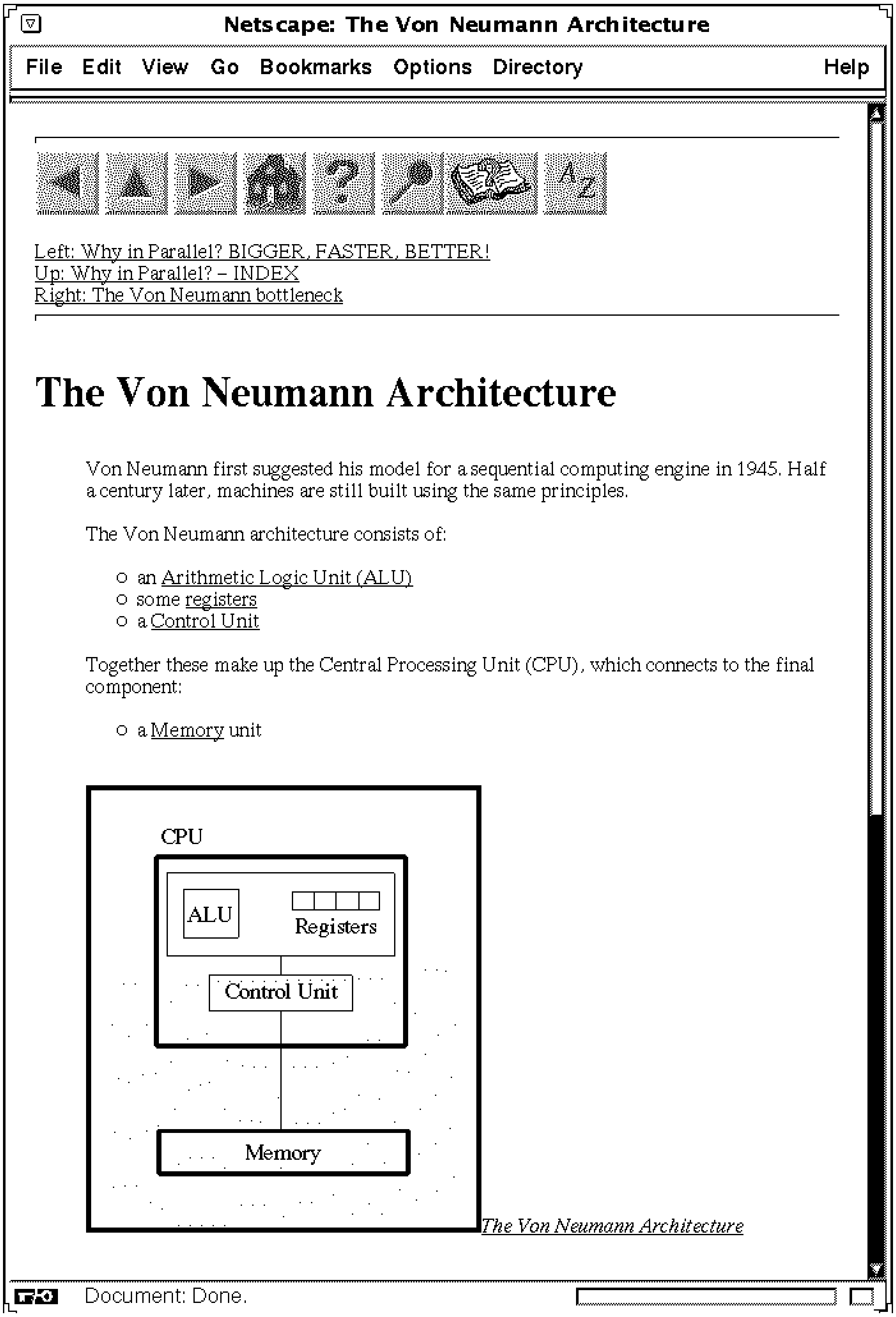


Figure 5: Example web node containing expanded **GLOSS** macros

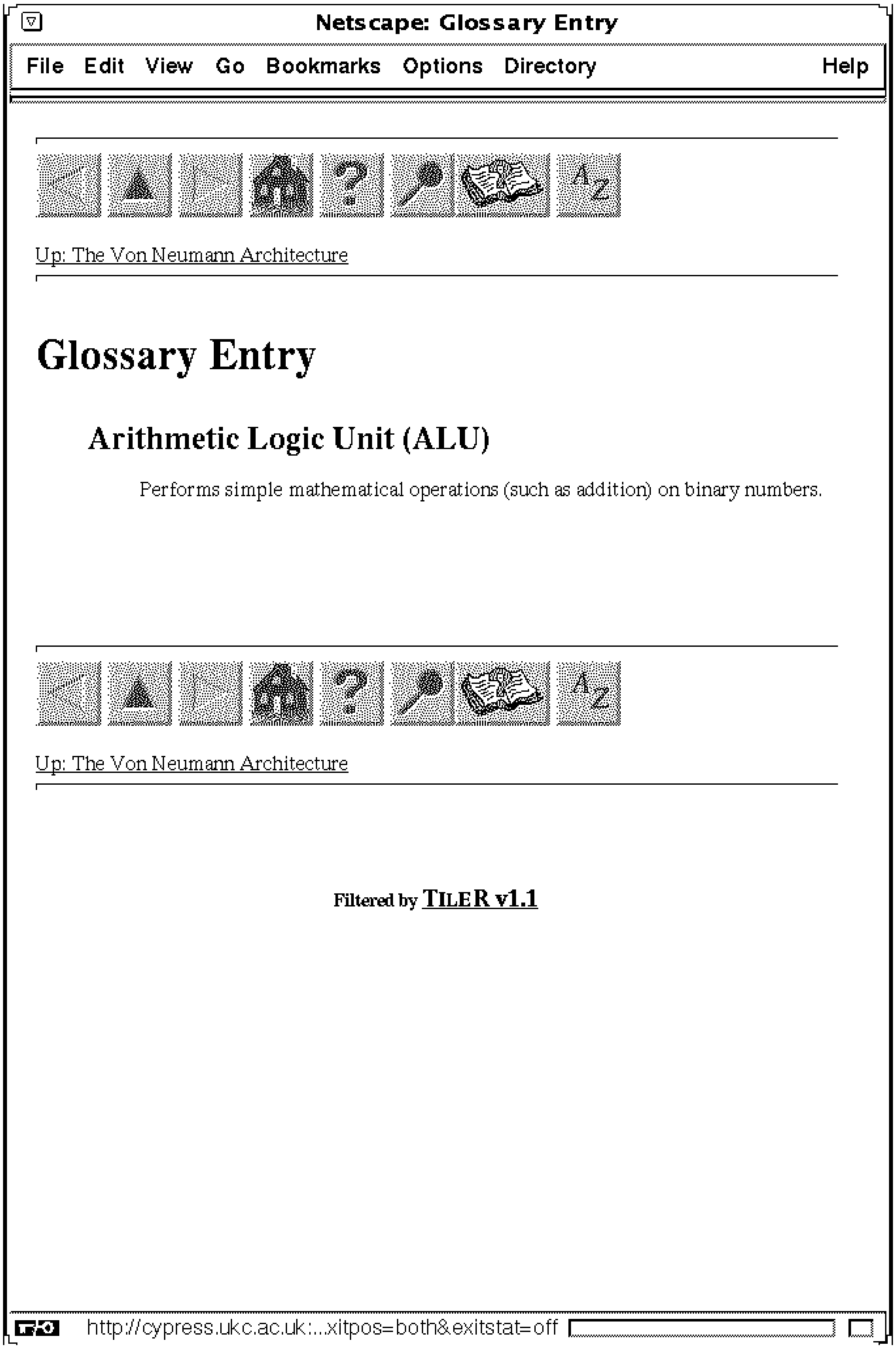


Figure 6: Example glossary entry containing further HTML and expanded macros

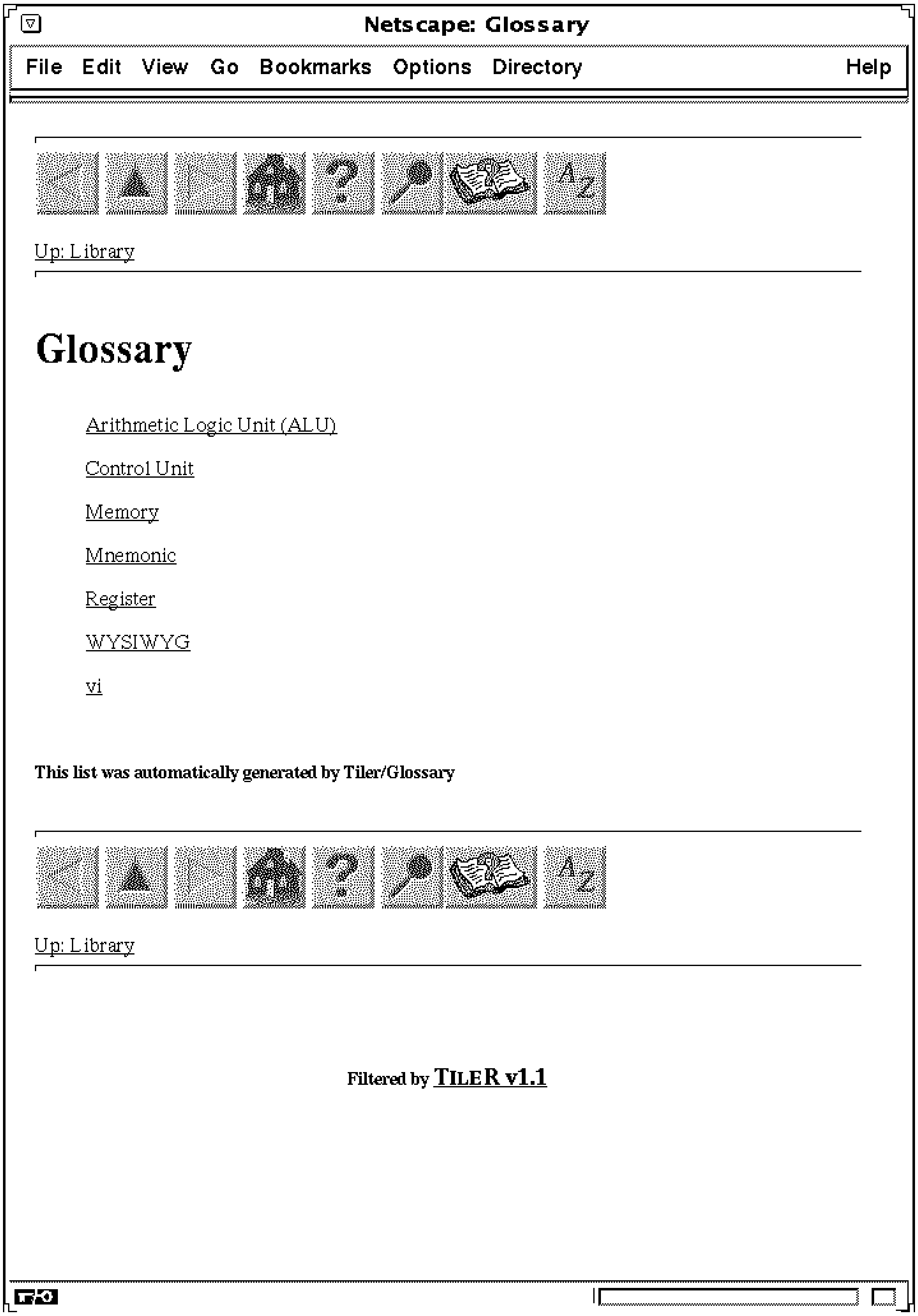


Figure 7: Example automatic glossary hyper-index

at the local server (see figure 8). As each new external page is filtered, encountered URLs are linked back to Tiler. Therefore the user can explore pages away from the local framework in the knowledge that one click of the emergency exit button can return them to where they left off.

9 The future 1: Softlinking v. Hardlinking

If each node in a web text contains a `BAR` macro specifying links to its parent and immediate siblings this still involves large amounts of repetitive manual markup. Also, moving a node to a different position in the web text requires not only altering the `BAR` macro in the node to be moved, but also the links to that node from other nodes.

An improvement on Tiler would be for information on the structure of the web to be stored in a central database. This is similar to the concept suggested by Berners-Lee [BL95] ("Some browsers have `next` and `previous` buttons to allow a document to be browsed serially") and implemented in Footsteps [NSS95], but by using two, rather than just one dimension, we can store hierarchical hypertexts as opposed to sequential trails.

For example, the structure of a web text could be stored in a single text file in the following format, with children listed in brackets after their parents:

```
dir/root.html {
     dir/sub-dir/child1.html
     dir/sub-dir/child2.html {
          dir/sub-dir/grandchild.html
     }
     dir/sub-dir/child3.html {
          dir/sub-dir/another-grandchild.html
          dir/sub-dir/yet-another-grandchild.html
     }
}
```

This would represent the following web text structure:

The advantage of replacing "*hardlinking*" with "*softlinking*" is that any alterations can be made to this single file. The resulting changes to links in individual web nodes can then be made automatically. This would further ease the maintenance of the web text whilst making it still more robust.

Softlinking also allows simple tailoring of courseware: any unwanted nodes can be removed simply by deleting their entry in the framework file; any extra nodes can be inserted by adding their entry. Once the central database is complete, the necessary updates to the linking of the individual web nodes can be made automatically.

10 The future 2: Pre-processing v. On-the-fly

In an ideal world, all macro expansion would be done on-the-fly, as with the Tiler macro processor system described in this paper. This is because only the original document need ever be stored, so saving disk space. But with limited processor speed available, it is impossible to deal with an ever increasing demand for web texts if the server has to expand macros before delivering them.

One solution would be to farm out the expansion process to the client. This would also result in less time being taken to deliver a document to the client, requiring less processor time,

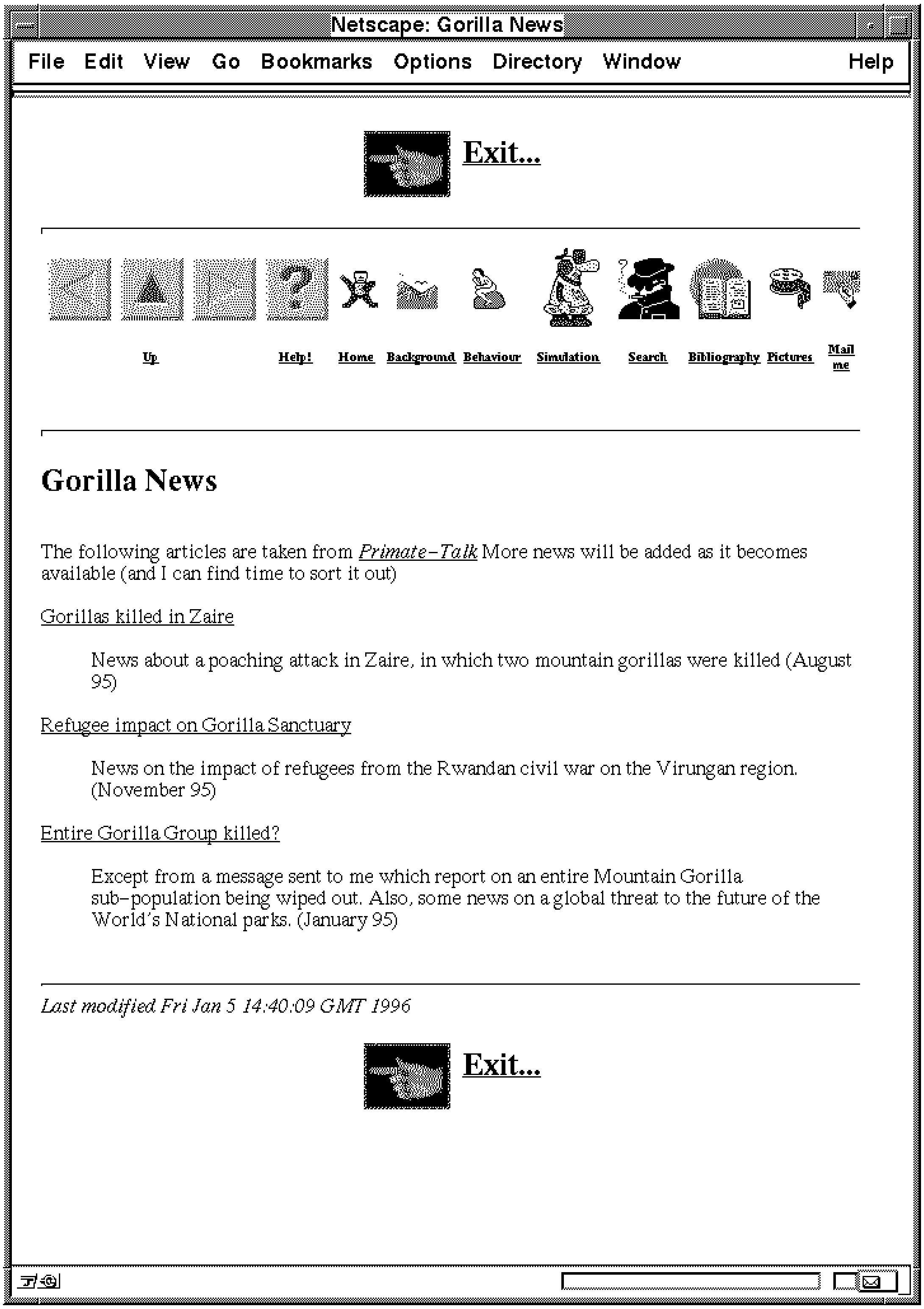


Figure 8: A filtered external page containing emergency exit buttons

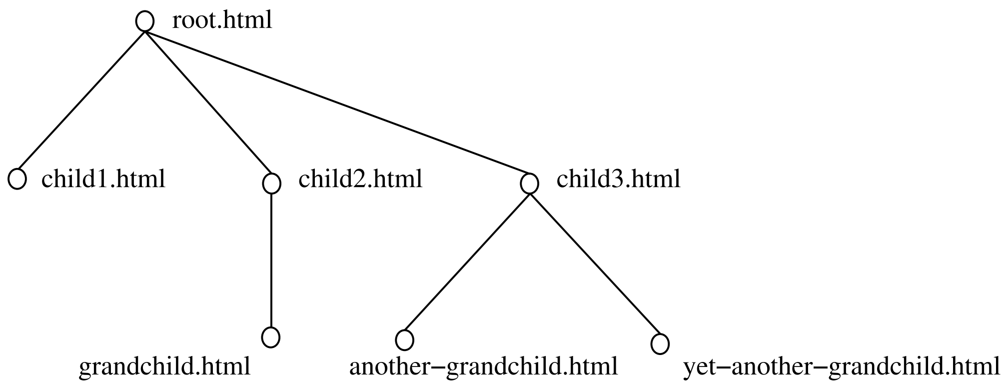


Figure 9: The Framework tree

less network bandwidth, and decreasing response times for the user.

Another possibility, which would be simpler to build from the current system, is to pre-process all files containing macros before they are served. This could be achieved using dual document trees: the first containing the source HTML and macros, and the second containing the expanded HTML resulting from pre-processing the source tree. The problem with this is that it requires at least twice as much disk space as the current on-the-fly method, as there are two versions of every web node (and the resulting web node is always going to be larger that the source web node).

11 Conclusions

The use of an on-the-fly mechanism is suitable for the time being, although busy web servers would suffer under the extra load. Another question mark in Tiler's usability has to be that all the macros mentioned in this paper are hardwired into the Tiler Perl source code; adding extra macros requires the user to be able to program in Perl.

However, the advantages heavily outweigh these disadvantages. The Tiler macro processor helps both users and developers in producing robust web texts. The biggest advantage of the Tiler concept is the decrease in manual HTML markup required by the developer to build sophisticated web pages.

References

[BL95] T. Berners-Lee. Style guide for online hypertext, 1995. <URL: http://www.w3.org/hypertext/WWW/Provider/Style/Overview.html>.

[BLCGP92] T.J. Berners-Lee, R. Cailliau, J-F. Groff, and B. Pollerman. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy, Vol. 1 No. 2, Meckler Publishing, Westport, CT, USA*, Spring 1992.

[Bro91] P.J. Brown. Hypertext: Dreams and reality. In H. Brown, editor, *Hypermedia/Hypertext and Object Oriented Databases*. Chapman & Hall, 1991.

[Dra94] N. Drakos. From text to hypertext: A post-hoc rationalisation of LaTeX2HTML. In R. Cailliau, O. Nierstrasz, and M. Ruggier, editors, *First International World-Wide Web Conference*, pages 179–190, Geneva, Switzerland, May 1994. <URL: http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html>.

[lib94] libwww-perl. Regents of the University of California, 1994.
<URL: http://www.ics.uci.edu/pub/websoft/libwww-perl/>.

[McC93] R. McCool. The Common Gateway Interface. NCSA, 1993.
<URL: http://hoohoo.ncsa.uiuc.edu/cgi/>.

[MP95] G.J. Makinson and A.T. Peel. High Performance Computing for all. In *The 3rd Annual Conference on the Teaching of Computing*, Dublin City University, Dublin, Ireland, 1995.

[Net95] Netscape Navigator Gold 2.0 data sheet. Netscape Communications, 1995.
<URL: http://home.netscape.com/comprod/products/navigator/version_2.0/gold.html>.

[NSS95] D. Nicol, C. Smeaton, and A.F. Slater. Footsteps: Trail-blazing the web. In *Third International World-Wide Web Conference*, Darmstadt, Germany, April 1995.
<URL: http://www.igd.fhg.de/www/www95/papers/60/footsteps.html>.

[Web95] K. Webber. Chapter 6, in which Pooh proposes improvements to web authoring tools, having seen said tools for the Unix platform. In *Third International World-Wide Web Conference*, Darmstadt, Germany, April 1995.
<URL: http://www.igd.fhg.de/www/www95/proceedings/papers/104/katew.paper.html>.

[WS92] L. Wall and R.L. Schwartz. *Programming Perl*. O'Reilly & Associates, 1992.