A Methodology for Structuring Web Documents

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}@cs.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

January 24, 1996

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 : Structuring, Automation, Navigation.

1 Background

The work described here was performed as part of the High Performance Computing Consortium Teaching and Learning Technology Project (HPCC TLTP[1]), a collaborative project involving the universities of Kent, Cardiff, Southampton, and London. The goal of this project is to provide hypertext undergraduate teaching material on High Performance Computing.

The first phase of this project was to examine and evaluate possible platforms for our material. After examining several hypertext packages, HTML and the world-wide web in conjunction with publicly available free web browsers were chosen as the most appropriate medium to use for producing this coursework. One of the main reasons for reaching this decision was the degree of platform-independance offered by the wide range of available web browsers and servers.

The second phase of the project is the production of coursework. This will include both interactive material (such as execution of example programs over the web and student programming exercises, both implemented using HTML forms) and non-interactive material (including text, graphics and animations). This phase will also include the development of support tools for the coursework.

2 A Structure for Courseware

It is possible to write course material in a standard wordprocessor or text-processing package (such as LaTeXor FrameMaker) and then use a tool such as LaTeX2HTML [1] or Webmaker [2] to convert it into web pages. However, we chose to write material directly in HTML in order to give us direct

[1] http://larch.ukc.ac.uk:2001/

control over the format and as this was seen as more appropriate for developing interactive material. Writing directly in HTML also allows us to view the final output without going through a conversion process, making development faster.

We agreed to a general principle that individual pages were to be kept small. This makes downloading individual pages fast and the user is not required to scroll through a large body of text. The drawback is that the user is obliged to navigate through a large number of pages. To prevent the user getting lost, we felt it was necessary to organize them into a structure that the user would be able to understand and utilize.

In writing material, the hypertext author is faced with a choice; either use links sparingly, so that the student has no choice but to read the material in the order chosen by the author, or use links extensively and risk having the student get sidetracked by material not directly relevant to the current topic of study. We concluded that the use of hypertext links by itself is not sufficient to impose a suitable structure on the set of HTML pages; the structure needs to be defined separately, so the student is aware of when they are following the prescribed material and when they are branching off to find out about a related topic.

We identified two distinct modes in which our courseware would be used and which the structure and links should support :

1. Tutorial mode. The student works through all the material in a section of the course. The pages should be encountered in a logical order, building on concepts and material that the student has already encountered and not using terms which have yet to be defined.
2. Reference mode. The user is familiar with the material, but wants to look up a specific point (possibly they are revising). It should be possible to get to any page with only a small number of intermediate pages. This can be achieved simply by generating an index of all pages, but we wanted it to be easy to locate a topic without resorting to an index.

We felt that the most appropriate model was a hierarchical (tree) structure. This combines flexibility with a structure that is reasonably easy to understand and traverse. Every page is a node of the tree and covers a topic. Some of the links on a page will be to child nodes; other pages which are sub-topics of that topic. At the top of the tree is a *root* page which covers all the topics within the scope of the course. We aimed for an authoring style that avoids "glue pages"; pages that do little more than list a set of sub-pages. Where such a page might be expected, we have an overview page which describes the material being covered with links to sub-pages generally appearing within this text.

Having seen the output from tools such as LaTeX2HTML and Webmaker, we liked the navigation systems provided and wanted to provide a similar navigation mechanism for our material. It seemed sensible to add this in an automatic or semi-automatic fashion using special tools. Our added navigation mainly takes the form of a "toolbar"; a series of clickable graphical icons on a single line. We call these clickable icons *buttons*. By using the buttons only, the student traverses the tree and can be confident of encountering the material in a logical order. Following links in the text allows the student the freedom to jump around between topics.

As hypertext authors, we also felt that it would be useful to have pseudo-HTML macros [3]. These are commands written in an HTML-like syntax which get converted into real HTML before the user's browser interprets them. They can simply be used as a shorter form of frequently used pieces of HTML or they may provide a means of referencing information only available at a later stage.

3 On-the-fly versus Static Pre-processing

3.1 When to Process each page ?

Each HTML page written by the author has a toolbar added, pseudo-HTML expanded and possibly other changes made before the user sees it. There are several ways in which this transformation could be effected :

1. Use a CGI script to add the toolbar *on-the-fly* as it is retrieved.
2. Modify every page *in-situ* once the course has been written.
3. Create a new version of the course by copying across and modifying (pre-processing) each page.

We rejected *in-situ* modification, because it destroys the original material, making it impractical to alter the course after the toolbars have been added. Both of the other two methods have their strengths and weaknesses. The advantages of both can be combined by implementing both and using pre-processing to deal with most special tags, but on-the-fly processing to handle features selected by the user at retrieval time.

3.2 On-the-fly Processing

In *on-the-fly* processing, all pages get processed by a CGI script at the Web server site before the user sees them. To access such a page, the user gives a URL which is the name of the CGI script to which is passed a reference to the page. If the GET method is used, the page reference can be appended to the URL, giving a URL of the form `http://machine/cgi-bin/script?page=pageref`. The script needs to make sure that future retrieved pages are also processed by the script. It does this by converting the HREF argument in every anchor tag on the page. If the URL is on the same server as the script, the URL can simply be converted to a `pageref` consisting of a path and a filename. If the URL is on a different server, it is still possible for the script to retrieve it from the remote site, for example using the `http'get` routine in the libwww-perl libraries [4]. In this case, `pageref` needs to be a full URL.

The advantages of on-the-fly processing are :

1. Only one version of the course need be stored.
2. It is possible for the user to select options such as which buttons are in the toolbar and whether the toolbar appears at the top or bottom of the page. More importantly from the educational point of view, it is possible to select options in the course material, e.g. languages and systems that the user wants to learn about, with each page returning only information relevant to those languages and systems.
3. It is possible to process pages returned from CGI scripts.
4. The HTML developer can immediately see each web page in its final form including the effect of any pseudo-HTML.
5. Pages cannot be cached, either by a proxy or by the browser.

However it also has disadvantages :

1. Every page retrieved needs to be parsed by the web server, possibly placing a heavy load on the server and hence reducing performance.
2. The mechanisms for user authentication provided by many Web servers no longer work, though it would be possible to build user authentication into the CGI script.

3.3 Static Pre-processing

In *static pre-processing*, the pages are copied to another directory and any pseudo-HTML tag expansion and addition of extra material is done at that stage.

The advantages of this scheme are :

1. The web server need only perform document retrieval, so there is no need to install a CGI script and loading on the server will be light.
2. The processing of URLs in anchor tags can be avoided; the HREF argument in anchor tags need not be changed if it is given as a relative URL (or if it is a reference to external material which is not to be processed).
3. The web server's user authentication scheme can be used.
4. Pages can be cached.

The drawbacks are :

1. The details of each page's format are settled when the course is set up and cannot take advantage of information known only at the time of retrieval. If we want to provide user-selection of course options, we have to do it by providing a copy of the document for each possibility. This becomes impractical once the number of possibilities grows beyond a handful.
2. The developer needs to perform a processing stage to see the final version of a page. However, since most Web browsers simply ignore tags that they don't understand, the developer can immediately preview all the normal HTML on a page.

4 Creating a Tailored Web Course

4.1 Overview

Once the course material has been written, the following steps need to be applied to create a tailored course on the web server :

1. Use an HTML search tool to extract the hierarchical structure.This will be represented within a single file, called a *structure file*.
2. Edit the structure file if desired.
3. Generate special pages such as contents, index and help.
4. Perform any pre-processing necessary. If no on-the-fly processing is to be performed, all the pages are copied to a new location and buttons are added. If an on-the-fly version is required, the pages are left where they are and the buttons will be added when the pages are accessed via a script. Possibly there is both pre-processing and on-the-fly processing. In that case the pages are partially processed by the pre-processor; they are copied to a new location and special tags are expanded but no buttons added. This reduces the work that the on-the-fly processor has to perform when the pages are accessed.
5. If required, install the on-the-fly processor as a CGI script.

4.2 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 were implemented as Perl programs. We believe that breadth-first search is the preferable strategy, since it minimizes the depth of the tree.

4.3 Adding Buttons

There was considerable discussion of which buttons should be included in the toolbar. Below we describe the buttons finally chosen.

Possibly most important button is the *up* button, which is linked to the current page's parent. The tree structure ensures that each page except the root page has a single parent page. With the addition of up buttons, it is possible to traverse the entire document from any node, not just the root.

Once each page has a unique parent, it also has a well-defined set of siblings. These are the other child pages of that tree's parent. These are given an order, by default the order in which they appear in the parent page. A *right* and a *left* button are provided to take the user to the next and previous siblings respectively.

The user will normally be able to access child pages by clicking on links in the text. However, some of the links may be pages which are not children (we refer to these links as *cross-references*). So that the user can be sure of visiting only child pages, we considered three mechanisms:

1. A *down* button which takes the user to the first child.
2. A list of child pages at the bottom of the parent page, with a link to each child.
3. A special icon next to each link in the text to indicate whether the link is a child link or a cross-reference.

After some discussion, we settled on option 2 as the most aesthetically pleasing.

In addition, we added :

- A *home* button, which takes you to the root page of the course.
- A *help* button, which takes you to a page describing the available buttons.
- A *contents* button, which takes you to an automatically generated contents page (with an entry for every page in the course).
- An *alphabetical index* button, which takes you to an automatically generated index page. This contains links to every page, but indexed on every word in every title.

4.4 Tailoring

Rapid development in the field of high performance computing and differing educational requirements of those establishments using our material make it desirable to support some degree of *tailoring* within our coursework. i.e. the ability to select pages and perhaps write new HTML pages for inclusion in the course. Thus a site which chooses to use our teaching material may set up a course using only a subset of the original pages. This is achieved by editing the structure file, allowing the following possibilities :

1. Removing unwanted pages from the course body. These pages are not physically deleted and are still accessible using cross-references in the body of the text. However, they cannot be reached using the added navigation buttons and will themselves have no navigation buttons added.
2. Re-ordering the children of a particular page so that the student encounters the topics in a different order.
3. Moving pages so that they are encountered at a different place in the course.
4. Adding new material that was not part of the original course.
5. Incorporating pages from other sites.

Once the structure file has been edited, another tool examines it to check the syntax before it can be used for building the course.

5 Pseudo-HTML macros

Whichever method of adding a toolbar is chosen, each HTML page is parsed before being presented to the user. This allows us to define our own set of pseudo-HTML macros which are converted into standard HTML. At its simplest, this allows us to use pseudo-HTML as shorthand for a frequently-used piece of HTML. Used in conjunction with information known to the parser or supplied as an argument, it can become more powerful. Among the special tags we chose to implement were :

- <IFDEF>, <ELSE>, <ENDIF> : For one course, C and Fortran versions of the material were required. The solution adopted was to create pseudo-HTML tags <IFDEF var>, <ELSE> and <ENDIF>. These are modelled on the #ifdef, #else and #endif constructs in the C pre-processor cpp. A small set of variables are declared to be "defined" and these are passed to either the static pre-processor or the on-the-fly interpreter. If, for example, C is defined, then everything between <IFDEF C> and <ELSE> is included in the final HTML, while everything between <ELSE> and <ENDIF> is omitted. If C is not defined, then only the material between <ELSE> and <ENDIF> is included in the final HTML. The <ELSE> is optional and if it is absent, the <ENDIF> behaves like <ELSE><ENDIF>. These constructs may be nested.
- <GLOSS> takes a KEY argument and a WORD argument and creates a hypertext link to a page explaining that word. The WORD argument contains the text that the user sees and can click on, while the KEY argument contains the key under which the appropriate entry is stored in the glossary database.
- <BULLET> takes a COLOUR argument (which is only allowed to be one of a fixed set of colours) and generates an in-line image of a bullet of the appropriate colour.

6 Current Status of Work

Figure 1: Example of a toolbar generated by Treetool.

The system described here arose through discussions among the authors of this paper and with the development of various prototype systems. As yet no individual tool exists which offers the full functionality described in this paper. We outline the prototype systems available, roughly in the order in which they were developed :

1. **Tiler** [3]. This uses the on-the-fly technique to support a variety of pseudo-HTML macros. It also supports the addition of buttons to pages stored at remote sites. Navigation is supported using a pseudo-HTML tag called <BAR> with arguments of the form BUTTONTYPE = URL. Each argument generates a button in the toolbar with a link to the specified URL. This approach was not adopted for our final system as it makes tailoring difficult and maintaining consistency of the navigation buttons requires some care on the part of the author.
2. **Grouter**. This again uses the on-the-fly technique to generate a tailorable course with a linear structure. The coursework is built out of units, each unit consisting of either a single page or a small tree of pages. The units are arranged in some linear order in a special file which can be edited. Traversal is allowed from each unit's root to the next and previous unit's roots. This makes is possible to step through each unit's root in sequence. We note that the result is similar to that achieved by the Footsteps tool [5]. Grouter also supports a number of pseudo-HTML tags.
3. **Treetool**. This takes a set of pages and constructs a structure file corresponding to those pages. However, the structure file is not intended to be edited. Other tools take the structure

file and use it to create a contents page and an alphabetical index. The course itself is built using pre-processing. Each page is copied across to another directory and a toolbar with up, right, left, help, top, contents and index buttons is added. The only pseudo-HTML tag supported is `<IFDEF>`. Figure 1 shows an example of the toolbar generated by Treetool.

4. **TLTPtool**. This uses the on-the-fly technique and structure files written by the user to create a tailorable hierarchically structured course. Each page has a set of buttons added and the user has some control over which buttons appear and where on the page the toolbar is. It also supports a large number of pseudo-HTML tags.

7 Acknowledgements

The system described here arose from discussions involving the authors of this paper. It was also influenced by earlier work within this project performed at Cardiff [6, 7]. This work is being carried out as part of the Training and Learning Technology Programme funded jointly by HEFCE, HEFCW, SHEFC and DENI.

References

[1] N. Drakos, "From text to hypertext: A post-hoc rationalisation of LaTeX2HTML," in *First International World-Wide Web Conference* (R. Cailliau, O. Nierstrasz, and M. Ruggier, eds.), (Geneva, Switzerland), pp. 179–190, May 1994. On-line documentation at http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html.

[2] B. Rousseau and M. Ruggier, "Writing documents for paper and WWW a strategy based on framemaker and webmaker," in *First International World-Wide Web Conference* (R. Cailliau, O. Nierstrasz, and M. Ruggier, eds.), (Geneva, Switzerland), pp. 169–178, May 1994. On-line documentation at http://www.cern.ch/WebMaker/.

[3] A. T. Peel, "HTML macros - easing the construction and maintenance of web texts," Tech. Rep. 4/96, University of Kent at Canterbury, Computing Laboratory, 1995.

[4] Regents of the University of California, "libwww-perl : Perl world-wide web library." available by ftp, 1994. On-line documentation at http://www.ics.uci.edu/pub/websoft/libwww-perl/.

[5] D. Nicol, C. Smeaton, and A. F. Slater, "Footsteps: Trail-blazing the web," *Computer Networks and ISDN Systems*, vol. 27, pp. 879–885, Apr. 1995. proceedings of Third International World-Wide Web Conference, held at Darmstadt, Germany. Available on-line at http://www.igd.fhg.de/www/www95/papers/60/footsteps.html.

[6] S. Hurley, A. D. Marshall, S. N. McIntosh-Smith, and N. M. Stephens, "Courseware for parallel computing using mosaic and the world wide web," in *Second World Wide Web Conference '94: Mosaic and the Web*, (Chicago, Illinois, USA), pp. 499–508, Oct. 1994. Available on-line at http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Educ/hurley/www.html.

[7] J. K. Campbell, S. Hurley, S. B. Jones, and N. M. Stephens, "Constructing educational courseware using NCSA mosaic and the world wide web," *Computer Networks and ISDN Systems*, pp. 887–896, Apr. 1995. proceedings of Third International World-Wide Web Conference, held at Darmstadt, Germany. Available on-line at http://www.igd.fhg.de/www/www95/papers/52/www3.html.