DOCUMENT MULTIVIEW USING CSS

Roger Stone

Department of Computer Science, Loughborough University, UK
r.g.stone@lboro.ac.uk

ABSTRACT

This paper introduces a lightweight, alternative delivery technique for web-based documents which contain information for various categories of reader. Information intended for the different categories of reader is identified by markup which exploits certain features of CSS, notably "display:none". The technique is simple to use and provides a client-side solution with dynamic filtering.

KEYWORDS

Navigation, web-documents, CSS, multiple-view, multiple-readers, dynamic filtering

1. INTRODUCTION

The premise of this work is that, despite all the modern navigational aids, there are still incompatible combinations of busy readers and unsatisfactory web documents where the reader feels there must be a better way of presenting the specific content that they want to read from a particular document. Typically a document will have information for several (many) different categories of reader. Depending on the document the reader categories can be related to different positions within an organisation, different areas of expertise, different levels of responsibility or different levels of interest. The challenge is to allow each different category of reader to 'see' the relevant parts of any particular document with least navigational effort on their part and without being distracted by irrelevant parts of the document.

2. BACKGROUND

There are various existing techniques and technologies that can be brought to bear on this problem. One obvious solution is to provide separate documents for each category of reader. This is a very attractive solution provided the information is extremely stable. But if the information is not stable, the effort required to guarantee correct simultaneous updating of multiple documents makes this solution unattractive.

If we deduce that it is better to keep just one master document, then the problem becomes one of providing a mechanism to deliver specific parts of the document to different categories of reader. Here there are several competing ideas. Conventional navigation partitions a document and allows the reader to jump into the document at a variety of different points. However the navigation menu in this case will be primarily created to express the skeleton of the document rather than the needs of particular categories of reader.

A reader can also attempt to find relevant content by in-document searching. This works best when the reader can express their interest in terms of keywords so that a search for these keywords separately or in combination reveals areas of interest within a particular document. There are two problems with this approach. One is that the reader's interest needs to be able to be expressed in terms of key words to find any information at all and the other problem is that relevant sections of the document may be missed. In any case the search typically throws up a

collection of starting points and the reader is required to manage the reading of the document from all of these starting points.

2.1. Reader-centred techniques

So far the discussion may be said to be about document-centred techniques. Among the reader-centered techniques is the idea of marking up portions of a document to show which categories of reader might be interested in certain parts. The intended consequence of this markup is that a filtered document can be supplied to each category of reader such that each reader starts with a reduced document containing only the material pertinent to their category. The reader can now read the whole document from beginning to end in the knowledge that all the content is relevant or they can browse their content using conventional navigational options.

There are two main options for the markup required for this technique in the world of web documents. One option is to invent a tag specifically for the purpose of identifying the class of intended reader so that arbitrary sections of the document can be enclosed with the new tag (e.g <reader category='manager'>…</reader>). If this is applied to an HTML or xhtml document this will obviously make it fail validation tests which may be unacceptable. However there are document authoring systems which contain this kind of flexibility (notably DocBook [1]) which can be used if it is acceptable to move the source document to this alternative form. The browser and the (human) reader need not be aware of these extensions and changes if the filtering is done as a server-side operation delivering valid HTML to the client.

A second option for the markup follows the ideas of microformatting [2] which, at its simplest, confers semantics via a class attribute. This confines markup to CSS styling which allows the full source document to remain HTML compliant.

3. NAVIGATION USING CSS

In what follows it is assumed that there are three different categories of reader which for the sake of generality will be referred to by the numbers 1, 2 & 3. The idea is to mark up the document using microformatting to indicate which categories of reader would be interested in any particular part. This will be indicated by using CSS styles *cat1*, *cat2* & *cat3*. The markup might look something like this:

```
<div>stuff for everyone</div>
<div class='cat1'>Stuff for category 1 readers</div>
<div class='cat2'>Stuff for category 2 readers </div>
<div class='cat3'>Stuff for category 3 readers </div>
<div class='cat1 cat2'>Stuff for category 1 and category 2 readers </div>
etc.
```

Now four separate stylesheets can be constructed which could be called *all.css*, *cat1.css*, *cat2.css* and *cat3.css*.

3.1. Alternate stylesheets

The separate stylesheets will be included in the document via links, not as four stylesheets all in force together but rather as alternate stylesheets, like this:

```
<!-- main default stylesheet -->
<link href=all.css' rel='stylesheet' rev='main' type='text/css' title=all />
<!-- cat1, cat2, etc. -->
<link href='all.css' rel='alternate stylesheet' rev='main' type='text/css' title='all' />
```

<link href='cat1.css' rel='alternate stylesheet' rev='main' type='text/css' title='cat1' />
<link href='cat2.css' rel='alternate stylesheet' rev='main' type='text/css' title='cat2' />
etc.

In addition a small script is required which can swap in a chosen stylesheet from among the alternatives [3]. Among the navigation provided in the document we anticipate including a reader-category menu with four entries (all, category1, category2, category3). It remains to design the menu and stylesheets so that a selection on this menu causes the document to be filtered as required.

3.2. The CSS style - display:none

In what follows the only aspect of the stylesheets discussed is the 'display' status, specifically 'display:none' which hides a section of the document. There will no doubt be other styles for visible content which influence such features as border, background, font, size, colour, amongst others. Our first attempt at defining the stylesheets for each category is:

File: cat1.css
```
/* display content suitable for category 1 reader */
.cat2 {display:none;}
.cat3 {display:none;}
```

File: cat2.css
```
/* display content suitable for category 2 reader */
.cat1 {display:none;}
.cat3 {display:none;}
```

These are curious stylesheets because the part that we are focussing on is not about establishing the font or colour or position or border of visible content but about hiding content from view. In fact the style used for a category1 reader (cat1.css) says "hide anything that is styled with cat2 or cat3".

Now the selector .x behaves in a way that is equivalent to

```
[class~="x"]      /* the attribute for class CONTAINS the substring x */
```

rather than

```
[class="x"]       /* the attribute for class EQUALS the substring x */
```

So clearly this over simplistic styling has the effect of hiding from a cat1 reader any sections that are styled with both cat1 and another category e.g. cat1 and cat2. So we have to add another rule to the end of cat1.css to over-ride the hiding effect of display:none.

File: cat1.css
```
.cat2 {display:none;}
.cat3 {display:none;}
div.cat1 {display:block;}
```

So for a section tagged * ... * the selector *div.cat1* over-rides the *display:none* matched by selector *.cat2* with *display:block*. Note that the div tag appears as part of the selector because the required attribute of the display property is different for different tags (a tag would require display:inline, a <tr> tag would require display:table-row, etc).

Although not implemented by CSS, a neater solution here would be if it was possible to allow the style display:*visible* meaning that the selected item is given its default visible styling.

If a (visible) styling is required that just applies to category1 readers then this can be applied by using attribute selection as follows

```
[class="cat1"] { background-color:red;}
```

This will match a section of the document marked up as <tag class="cat1"> but will not apply to a section marked up as <tag class="cat1 cat2">. If a mixture of these styles is being used then it may be necessary occasionally to write something like <tag class="cat1 cat1">... . This will then be matched by the .cat1 selector but not by the [class="cat1"] selector.

Thus we have a basic system for filtering the content down to that relevant to one of a predetermined category of readers. When in use the overall effect is quite similar to the use of an document outlining system where menu selections hide and show parts of the document. The difference here is that the parts of the document that are hidden or shown are not hierarchically related.

3.3. Granularity of markup

So far the granularity of the markup has not been discussed. It might have been assumed that only block level components of the page (e.g. , <p> tags in markup terms or chapters, sections, paragraphs in document terms) could be marked up in this way with the browser being able to hide them neatly during dynamic rendering leaving the display tidy and the reader unaware of 'missing' content. In fact the technique can be used down to the level of individual characters and it works surprisingly well with table cells. Suppose that information is being presented in a table for the benefit of different categories of reader.

Feature	Category1	Category2
Feature 1	√	
Feature 2		√
Feature 3	√	√

Suppose the following styles are in place

Column Category1: cells are styled with class="cat1"
Column Category2: cells are styled with class="cat2"
Row Feature 1: row is styled with class="cat1"
Row Feature 2: row is styled with class="cat2"
Row Feature 3: row is styled with class="cat1 cat2"

Then for if a reader changes the viewing selection from 'all to 'category 1', the browser will be requested to activate stylesheet cat1.css, the document will be re-rendered dynamically by the browser and the table shrinks to

Feature	Category1
Feature 1	√
Feature 3	√

3.3. Multi-dimensional categories

So far it has been assumed that there is only one dimension to the categories. However it is quite common for there to be orthogonal categories or multiple dimensions (e.g. a student would normally be regarded as being in a certain programme category and in a year category). The only extra arrangement needed to allow these orthogonal categories is to make corresponding independent groups of alternative stylesheets. When a category selection is made the style switcher has to disable the currently active alternate stylesheet in that group and enable the chosen one. In the current implementation the alternate stylesheets are grouped by the *rev* attribute and this provides the only extra information that the switcher needs.

```
<!-- programme: CS|CSEB|CSAI|CM|ITMB|ITM -->
<link href='styles/CS.css' rel='alternate stylesheet' rev='programme' type='text/css' title='CS' />
<link href='styles/CSEB.css' rel='alternate stylesheet' rev='programme' type='text/css' title='CSEB' />
etc.

<!-- part: partA|partB|partI|partC|partD -->
<link href='styles/partA.css' rel='alternate stylesheet' rev='part' type='text/css' title='partA' />
<link href='styles/partB.css' rel='alternate stylesheet' rev='part' type='text/css' title='partB' />
etc.
```

4. EXAMPLES

4.1. Essentiality markup for the visually impaired

The ideas presented here were developed initially within the context of work done to improve the accessibility of web pages for readers whose impairment was such that they could not cope with large amounts of information. For web users who need to use screen magnifiers or who need to use a screen reader the concept of essentiality markup was explored [4]. Following this idea a document would be marked up by the author in various levels of essentiality and by referring to a reader profile, a proxy server would deliver a version of the document reduced in size to make it acceptable to the reader [more detail in section 6]. An unexpected outcome of this work was the comments of people without a disability who came into contact with the research who made remarks like "that would be of use to me".

4.2. Programme regulations

The effect of the technique is well demonstrated by the departmental programme regulations example [5]. In this example the categories are represented by students on six different programmes. At one time when there were less programmes and less similarity the programme regulations were kept in separate documents. The arrival of more programmes and greater similarity was the inspiration both to use a single document and to look for a natural way for the students to navigate the document. A further orthogonal categorisation in this example is the five possible parts (or years) of their programme. So this example supports a two dimensional grid of categories (programme, part). Each of the programmes is given a separate distinct colour. The parts of the document which give information for a single programme only are colour coded with the appropriate colour. This was a big help to the academic staff interpreting the document to other staff and students. However it was felt that it was important to allow the students to obtain a version which was equivalent to the separate, programme-specific document that had been provided in the past.

4.3. WCAG

Another piece of work centres around the document known as WCAG 1.0. (Web Content Accessibility Guidelines, version 1) [6,7]. It is argued that all stakeholders in a website should be aware of this document to ensure that the eventual website is accessible. However the level of detail required by the eventual customer and those involved in analysis, specification and

coding are very different and this gives one dimension of markup category. The guidelines offer advice about different kinds of disabilities (principally visual, auditory, cognitive, motor) and therefore the content relating to separate disabilities provides a second dimension of markup category. It is generally thought that a web document has the three components namely content, navigation and user-interface. This provides a third dimension of markup. The document is structured into 14 guidelines with three levels of detail so these provide another two dimensions of markup. The original W3C document has been marked-up for these five dimensions so that by selecting (e.g.) stage|analysis and disability|visual the document should be reduced to what an analyst should know about maintaining accessibility for visually impaired visitors to a site. This marked-up version of WCAG is being trialled experimentally with real people involved at different levels with web-site production to try to measure the effectiveness of the technique. The 5-dimensional navigation menu is shown below.

Stage	**Disability**	**Cont-Nav-UI**	**Level**	**Guideline**
All	All	Content	Detail1	1
Analysis	Visual	Navigation	Detail2	2
Specification	Audio	User-interface	Detail3	3
	Cognitive			:
	Motor			14

5. CROSS BROWSER COMPATIBILITY

A major problem with the technique as presented so far is that while browsers such as Firefox, Opera, Safari, Camino manage to render the documents as expected, there is a problem with Internet Explorer. IE6 is the first browser from Microsoft to be able to handle the attribute selection classes used above. However it does not seem to be able to handle the critical situation described above where there is a stylesheet containing display:none immediately overridden by display:*something-visible*. The desire to provide cross-platform, cross-browser compatibility for this technique provided the impetus to search for the alternative method which is described next.

An alternative solution was sought that did not require display:none to be overridden. Suppose we have classes A,B,C,D corresponding to 4 categories of viewer. We can choose to style a section for multiple categories (say A and D) by

```
<span class="A B-no C-no D">...</span>
```

and a section for a single category (say A) by

```
<span class="A">...</span>
```

If we have viewer of category A, then we switch to stylesheet

File: A.css

```
[class="B"]{ display:none;}
[class="C"] { display:none;}
[class="D"] { display:none;}
[class~="A-no"] { display:none; }
```

i.e. anything just styled for B or C or D will be hidden. Anything styled for a mixture of A,B,C,D will be hidden if it contains A-no. This works successfully across Camino, Opera, Safari, Firefox *and* IE. This method has been applied to the student regulations example.

6. NESTED CATEGORIES

Now that the technique is working in a broad range of browsers, the question might be asked as to whether there are there any other limitations. One awkward area relates to "nested markup", specifically where for a certain category of reader a portion of content that should be visible to the reader appears inside a larger portion that should be invisible.

To give a specific example [see also 4.1], suppose the essentiality concepts ess1, ess2,…,ess10 (ess1=least important, ess10=most important) are being used and the document is marked up with nested portions like

```
<p class="ess1">Boring preamble
        <span class="ess10">DON'T MISS THIS</span>
                back to the boring stuff again</p>
```

If the document did not contain any nested category portions like this a stylesheet to see the document at essentiality level 5, could simply be

```
File: ess5.css
.ess1, .ess2, .ess3, .ess4 {display:none;}
/* the expectation is that content with class ess5, ess6, … ess10 is visible */
```

However with the nesting of categories this has unexpected results. In the example above viewed in category ess5 (stylesheet ess5.css) the paragraph style (class="ess1) is matched and instantly the whole of the content of the paragraph (including the nested tag content) is hidden. When it comes to the nested none of the browsers seem to be able to override the display:none style established by an outer tag and so the nested portion is not made visible.

There are two fairly unsatisfactory solutions to this problem. The first is to give up using display:none and try something like color:white to make some writing 'invisible' – however this can produce long, mysterious 'blank' sections in documents. The second solution is to change the markup so that it is not nested, however this is quite hard work in practice.

```
<p>     <span class="ess1">Boring preamble</span>
        <span class="ess10">DON'T MISS THIS</span>
        <span class="ess1">back to the boring stuff again</span>        </p>
```

There is a further solution to the problem and that is to use XSLT for the styling rather than CSS. In this case the source document will have to be well-formed in the XML sense (e.g. XHTML). The essential concept used in the stylesheets changes from display:none to that of copying a node (or not). Thus for example an XSLT stylesheet might be prepared as follows

```
<!-- If the class mentions cat2 then the node must be copied -->
<!-- e.g. class='cat2 cat3'  or class='cat2' -->
<xsl:template match="*[contains(@class,'cat2')]">
        <xsl:copy>
                <xsl:apply-templates select="."/>
        </xsl:copy>
</xsl:template>

<!-- if the class mentions a category other than cat2 then the node must be omitted -->
<!-- e.g. class='cat1 cat3'  or class='cat3' -->
<xsl:template match="*[(contains(@class,'cat1') or contains(@class,'cat3')) and
```

```
                                        not(contains(@class,'cat2'))]">
        <xsl:apply-templates select="."/>
</xsl:template>

<!-- otherwise nodes are copied -->
<xsl:template match="@*|node()">
        <xsl:copy>
                <xsl:apply-templates select="@*|node()"/>
        </xsl:copy>
</xsl:template>
```

The nice thing about XSLT is that the stylesheets can originate as strings which means that they can be constructed algorithmically on-the-fly. There is no need to prepare in advance a large collection of similar, but critically different, stylesheet files. The disadvantage with XSLT is that the various browsers cannot be relied upon to perform this styling on the client side, so we are forced back to a server side solution.

There are some deep problems associated with multiple markup of documents where the alternative markups exhibit different hierarchies [8,9]. This is a separate research strand and is beyond the scope of this paper.

7. TOOLS

The markup advocated in this document is very easy to understand but would be tedious to apply using a plain text editor. The use of a WYSIWYG web page editor helps greatly. For example in Dreamweaver [10] it is possible to select a vertical column in a table and apply a style to the whole column in one operation. However even this high quality software tool has its limitations relating to the styles used in this paper. One limitation stands out and that is that Dreamweaver (in WYSIWYG mode) has no way of applying multiple classes to a single element, i.e. two or more style names in a class attribute (e.g. class="cat1 cat2"). Thus these styles must be typed in text mode where needed.

The design of any special purpose tool to create and apply the styles required would need to take into account the repetitive nature of the alternate stylesheets required and generate them automatically from minimal information. The tool should also provide the author with a convenient way to cope with the (unusual) emphasis on multiple class styling.

8. CONCLUSIONS

A lightweight, client-side method of offering multiple views of an HTML document has been demonstrated which relies on the use of certain (less common) properties of CSS styles. By creating alternate stylesheets and by having navigation menus linked to a stylesheet-switcher script, different categories of readers can select different views of a document. The chosen stylesheet will cause certain portions of the document to be hidden using the CSS style display:none. In response to a category selection by a reader the browser's rendering engine will re-style the document dynamically. The categorisation can be multi-dimensional if required. Care must be taken to ensure that the stylesheets are cross-platform and cross-browser compatible. The amount of a document that can be styled for a particular category can vary from a single character up to sections containing multiple paragraphs. The technique works surprisingly well on table structures.

The key feature of this technique that sets it apart from other navigational systems is that it filters the document so that the only visible parts are relevant to a particular category of reader. This means that the reader seems to be getting a purpose written document for their specific

reader category. However this is obtained from a single source and thus avoids the difficult update problem associated with multiple documents. There are problems associated with documents that would naturally be marked up with nested categories. A solution for this situation is included.

ACKNOWLEDGEMENTS

The author would like to acknowledge the participation of PhD students Matthew Atkinson, Jatinder Dhiensa and Rehema Baguma in aspects of this work.

REFERENCES

[1] DocBook (a semantic markup language for technical documentation), http://www.docbook.org/

[2] Microformatting, Adding information to a web page using mostly the class attribute, http://microformats.org/get-started/

[3] Stylesheet Switching tutorial, http://www.alistapart.com/stories/alternate/

[4] Dhiensa, J., Machin, C.H.C. and Stone, R.G., "Creating a User-specific Environment: The Author Implications", Proceedings of the IADIS International Conference, WWW/Internet 2006, Isaias, P., Nunes, M.B. and Martinez, I.J. (eds), IADIS Press, Murcia, Spain, 2006, 235-242, ISBN 9728924194

[5] Departmental Regulations, Computer Science, Loughborough University, https://co-public.lboro.ac.uk/corgs/regs/regs2008/ug-regs/ug-regs.htm

[6] Web Content Accessibility Guidelines version 1.0 (WCAG), http://www.w3.org/TR/WCAG10/

[7] Baguma, R., "A Framework for Filtering Accessibility Guidelines", submitted to the International Cross-Disciplinary Conference on Web Accessibility 2009 (W4A)

[8] Cristea, D., Ide, N. and Romary, L., "Marking-up Multiple Views of a Text: Discourse and Reference", Proceedings of the First International Conference on Language Resources and Evaluation, Granada, Spain, 1998, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.1004

[9] The Text Encoding Initiative (TEI), Multiple Hierarchies, http://www.tei-c.org/Guidelines/P4/html/NH.html

[10] Adobe Dreamweaver, web authoring tool http://www.adobe.com/uk/products/dreamweaver/