Automatic Pagination of HTML Documents in a Web Browser

Alfie Abdul-Rahman, Roger Gimson, John Lumley

HP Laboratories
HPL-2009-123

Keyword(s):
XSLT, DOM, adaptive layout, pagination

Abstract:

A typical HTML document that is viewed in a web browser usually requires some form of scrolling actions from the user in order to view the whole document. In this paper, we present an application for the automatic pagination of an HTML document in a web browser, based on the dimensions of the browser window. Our application does not involve any server-side or proxy technologies or any additional plug-in or code beyond the JavaScript and XSLT support in the browser. It is based on a two-stage process - (i) decoration of the HTML Document Object Model (DOM) with its coordinates and sizes using a library of JavaScript functions and (ii) pagination of the HTML DOM using a transform via the browser built-in XSLT 1.0 processing engine.

External Posting Date: July 24, 2009 [Fulltext]
Internal Posting Date: July 24, 2009 [Fulltext]

Approved for External Publication

Automatic Pagination of HTML Documents in a Web Browser

Alfie Abdul-Rahman, Roger Gimson, John Lumley
Hewlett-Packard Labs
Long Down Avenue
Bristol BS34 8QZ, UK
{alfie.abdul-rahman, roger.gimson, john.lumley}@hp.com

ABSTRACT

A typical HTML document that is viewed in a web browser usually requires some form of scrolling actions from the user in order to view the whole document. In this paper, we present an application for the automatic pagination of an HTML document in a web browser, based on the dimensions of the browser window. Our application does not involve any server-side or proxy technologies or any additional plug-in or code beyond the JavaScript and XSLT support in the browser. It is based on a two-stage process – (i) decoration of the HTML *Document Object Model* (DOM) with its coordinates and sizes using a library of JavaScript functions and (ii) pagination of the HTML DOM using a transform via the browser built-in XSLT 1.0 processing engine.

Categories and Subject Descriptors

I.7.5 [**Computing Methodologies**]: Document and Text Processing– Document Capture[document analysis]

General Terms

Algorithms, Design

Keywords

XSLT, DOM, adaptive layout, pagination

1. INTRODUCTION & MOTIVATION

This paper describes some of our research on adaptable presentation of documents. The issues that are being explored include device independent authoring of documents, such as HTML pages. A significant amount of research effort has been directed towards achieving device independence, in particular the rendering of web pages on small screen devices [4]. The process of rendering the pages typically involves the analysis of the structure and layout of the document and the delivery of its presentation to the device either using client-side methods, server-side technologies or a proxy [6].

.

The main motivation behind this paper is to explore the possibility of implementing a client-side approach of automatic pagination of HTML documents based on the dimension of the web browser's window. An advantage of implementing such an approach client-side is that a human viewer of the document can be presented with an alternate presentation of the document without the additional communication cost between the server (or a proxy) and the user's machine. Also, it is easier to make use of the built-in rendering abilities of the client during pagination.

A number of studies have been carried out comparing the usability of paging and scrolling. The issues have been explored from several view points which range from the effectiveness of displaying search results [3] to the reading and understanding of text [1]. Which one of the two approaches is preferable to the human viewer depends on the nature of the HTML document.

The remainder of this paper is organized as follows. In Section 2, we describe related work. In Section 3, we describe our proposed application of automatic pagination of the HTML documents together with some example results and possible extensions for the application. Finally, in Section 4, we provide our observations and concluding remarks.

2. RELATED WORK

An HTML document is usually considered to represent a single web page.

The pagination of an HTML document involves partitioning the document's content and presenting it on individual pages (or *sheets*). The W3C's CSS Level 3 Working Group has proposed some mark-up for web page pagination using *Template Layout Module* [10] and *Paged Media* [11]. An example of a Paged Media module implementation can be seen in *Prince* [12].

Other discussions on pagination of HTML documents can also be found on the web. Examples include the pagination of HTML documents based either on cut-off markers or the number of items to be displayed per page. These examples can then be implemented either using server-side technologies [2, 7] or client-side methods [8]. These approaches are different from our proposed application, which is a client-side implementation where the pagination is based not on the document logical structure but on the dimension of the web browser's window and the rendered size of components.

We do not explore the usability issues that may arise from the pagination of HTML documents. Baker [1], Bernard *et al.* [3] and Peytchev *et al.* [9] amongst others consider this issue.

3. DESCRIPTION

Our proposal for automatic pagination of an HTML document is a client-side implementation, composed of two parts – (i) a library of JavaScript functions for sizing the pages and decorating the HTML DOM with size attributes and (ii) a pagination process via an XSLT transformation. The overall architecture is shown in Figure 1.

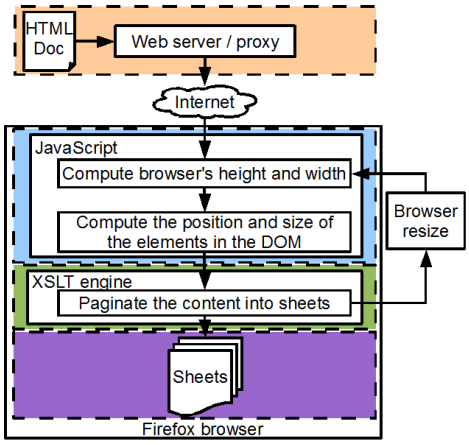

Figure 1: Overall architecture of the automatic pagination application.

The browser is instructed to perform pagination by embedding a library of JavaScript functions in the `head` tag, invoked via `onload` and `onresize` event-handlers. `onload` is initiated when the HTML document is first loaded to the web browser, while `onresize` is invoked each time the browser's window size changes.

The elements (or content) that are to be processed are located in the `body` of the HTML document. These elements are typical HTML nodes and will be transformed later on in the pagination process.

Document model: In our implementation, we assume that the layout in the document has not been arranged by absolute positioning through the CSS position property and the leaf nodes are considered to be atomic pieces in flow order. That is the elements are read from top to bottom of the page and arranged one after another. As the browser's window is resized, it is assumed that the browser's layout engine will re-arrange the elements accordingly. As an HTML document contains a hierarchical structure in its nested tags, the structure can be parsed as a tree of objects in the DOM.

Detecting & recording geometry: Given such a document, we first traverse the HTML DOM tree using a library of JavaScript functions. These functions decorate each element in the `body` node with its coordinates and size as attributes, which record approximate space that each element occupies in the browser's window. These coordinates and sizes are re-computed each time the browser's window changes. The two attributes that are important in our implementation are the y-coordinate and height of the elements (see Figure 2). We define this decorated DOM tree as a pre-process DOM tree and keep an invisible copy of it by setting its `display` attribute to *none*. By keeping a copy of the pre-process DOM tree, we are able to maintain the original structure of the HTML document and use it each time the browser's window is resized (see Figure 3).

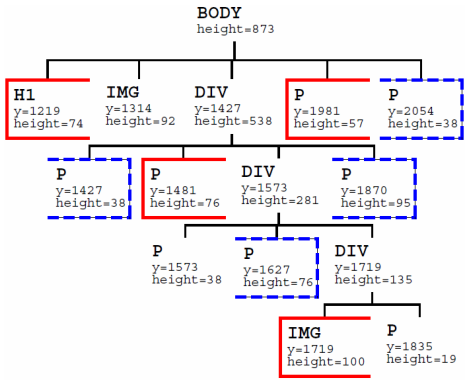

Figure 2: Structure of pre-process DOM tree of the HTML document shown in Figure 4(a).

Splitting to sheets: The decorated DOM is then passed to the second part of our application – a paginator, implemented in XSLT. Figure 2 shows an example structure of the pre-process HTML document of Figure 4(a), where solid and dashed boxes indicate the start and end leaf nodes of each sheet in a depth-first traversal of the tree. The `XSLTProcessor` object[1] in JavaScript defines a `Transformer` class that can encapsulate an XSLT stylesheet and apply it to the 'original' HTML document. The transformer then paginates the DOM into several sheets. The overall size of the content in each of the generated sheets is the approximate size of the dimension of the web browser's window. The resulting sheets are then re-inserted into the DOM by the `Transformer` class and later rendered by the browser's layout engine. Figure 3 shows the resulting paginated DOM tree of the HTML document together with the 'hidden' copy of the original input.

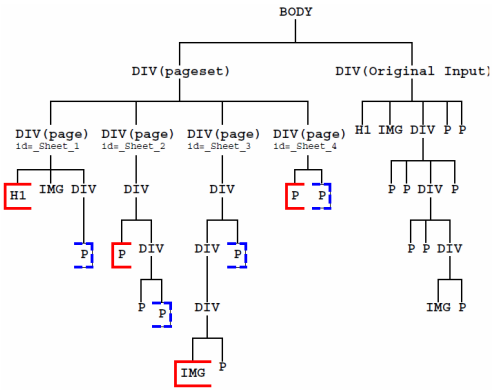

Figure 3: Structure of resulting processed DOM tree of the HTML document as well as its 'hidden' original input shown in Figure 4(b).

The results of the pagination together with its unpaginated version can be seen in Figure 4. Figures 4(b) and (c) show that the content and structure of the HTML document are maintained by our application (*i.e.,* no information is added or lost during the decoration and pagination processes). The only object that was appended to the document is the navigation mechanism located on the top of

[1]In Internet Explorer, a `transformNode()` method is used.

(a) (b) (c) Sheet 1 – Sheet 4

Figure 4: (a) An unpaginated version of the HTML document. (b) Showing where the pagination breaks will be inserted. (c) The results of the paginated document as viewed in the browser, with added navigation.

each sheet.

Although our application is able to handle most tree structures and CSS styling, there are cases where further work is needed, such as, changed sibling positions affecting the numbering of ordered lists and preserving position sensitive CSS styling.

As the pagination process of the sheets is implemented via the built-in XSLT processing engine, the HTML DOM must be well-formed. Like many current web pages it requires that JavaScript is enabled in the web browser. Our approach has only been implemented using the Firefox browser but we believe that it can also be implemented on other browsers that support the manipulation of the DOM. We have implemented our application using a combination of JavaScript functions and XSLT because required tree manipulation is more succinctly defined in XSLT. However, there is nothing to prevent re-implementing the whole application in JavaScript.

3.1 Implementation Details

In this section, we discuss in further detail the two main components of our pagination application.

3.1.1 *Decoration of the HTML DOM*

The attributes that are added as decorations on the HTML DOM tree, defined using JavaScript properties[2], are as follows:

***x* and *y*:** specify the coordinates of the elements in the DOM and are determined from the `offsetLeft` and `offsetTop` properties.

***width* and *height*:** defines how much space is occupied by the element and are computed from the `offsetWidth` and `offsetHeight` properties.

In addition, the dimensions of the browser's window (*i.e.,* *width* and *height*) are also determined and are added as attributes to the top of the DOM tree. The *width* and *height* are calculated from the `innerWidth` and `innerHeight` properties of the window object.

All the above attributes are annotated onto the HTML DOM tree using JavaScript. This DOM tree is then hidden by setting its `display` attribute to *none*.

3.1.2 *Pagination of the HTML DOM*

Now we have all the necessary geometry, pagination involves two parts:

Dividing elements into sheets: We first determine the area that is available for display by taking into account the amount of space that is occupied by the navigation mechanism. Each of the elements in the DOM tree is labelled with a unique identifier that is used for identifying the start and end leaf nodes in each group. We assume that the leaf nodes are atomic pieces in flow order.

Beginning with the first element in the DOM tree, an element is considered to be part of a group if its *end-y* (*i.e.,* sum of its *y*-coordinate and height) is less than the displayable height that was calculated earlier. This process is repeated until we have reached the last element in the DOM tree. There will be situations where the height of an unbreakable element is greater than the displayable height and, for the moment, the element will be displayed as it is. There are several improvements that can be implemented depending on the element's type. For an image, we can reduce the size of the image to fit the displayable area and for a text paragraph, we can split the paragraph into lines and choose which point to perform the pagination.

Gathering elements into sheets: Using the unique identifiers of the start and end nodes in each group, we then traverse through the DOM tree and copy the nodes that are located within the start and end nodes, placing each group of nodes within an appropriate `div`. It is vital that we maintain the tree structure and preserve the ancestry of the leaf nodes as CSS styling of the document can be affected by them and this may involve the duplication of intermediate ancestor nodes that 'straddle' a page boundary.

3.2 Navigation of the Sheets

Rather than using embedded *back* and *forward* buttons

[2]These properties are specific to Firefox and may be different for other browsers.

in the page, in this implementation our navigation tool for manoeuvring between the sheets is a horizontal tab menu that shows the sheets that can be found in the document. The tab menu is created using a combination of HTML, CSS and some JavaScript functions. This approach of building a horizontal tab menu is similar to that discussed by Blixt [5].

Each of the tabs represents a paginated sheet. The tabs and divisions are dynamically generated during the pagination process and the resulting tab menu and sheets are inserted into the document. Each of the sheets is made visible by JavaScript functions, which modify its `display` attribute to either *none* or *block*.

One of the advantages of choosing and designing this form of navigation is that we do not need to create any hyperlinks and no additional page needs to be created for each paginated sheet, allowing everything to be done client-side. However, a disadvantage is that the tab menu occupies space on the sheet and must be a constant size.

3.3 Possible Extensions

In this section, we discuss some of the possible extensions that can be implemented in our application, either in the paginator or embedded in the HTML document.

3.3.1 Paginator

Our application is able to automatically paginate HTML documents that contain both text and images. The pagination is performed between nodes. However, there will be cases where it is necessary to split a paragraph into lines and decided at which point between the lines to perform the pagination. This extension is very desirable in our application but may not be implementable in current browsers that are unable to retrieve information below the DOM level. In order to implement this, better access to the browser rendering engine is needed.

3.3.2 HTML Document

There are various ways of structuring the HTML documents for pagination. Early on in the paper we have shown how a basic model of an HTML document can be paginated. We can extend this model by additionally representing a footer and a header within the document. This form of structuring should be easy to implement and it also avoids the task of analyzing the document in order to identify its semantic structure.

In the course of paginating an HTML document, there will be situations where it is desirable to provide alternative content for a specific element. This could be due to either the limited space of the displayable area, restriction based on the device capabilities or user preferences. For example an image can be displayed at its full resolution, at a reduced resolution or by providing only a description of the image.

Some other extensions that can be embedded inside the HTML documents include:

Logical grouping of content: Content can be logically clustered according to specific groups as a way of describing their connectivity. For an example, a particular piece of a paragraph may be grouped together with an image therefore forcing them to be paginated onto the same sheet.

Cut-off markers: Cut-off markers may be included inside an HTML document, indicating the desirable pagination points and including additional constraints to the pagination process.

4. CONCLUSIONS

So far the work in this paper has only explored a small area of adaptable presentation of documents, and there is much further work that can be done. In this paper, we have reached the objective that we set out to achieve:

- Designing and implementing a client-side application that enables the automatic pagination of an HTML document based on the dimension of the web browser's window and the rendered size of components.

We are able to provide an alternative paginated means of presenting an HTML document. Furthermore, by performing the pagination client-side, we have been able to save the communication cost between the server (or a proxy) and the user's machine.

5. ACKNOWLEDGMENTS

The authors would like to thank Tony Wiley for support to this work, and Owen Rees who provided helpful discussions of this work and his assistance in bug tracking in the initial implementation of the application.

6. REFERENCES

[1] J. R. Baker. The impact of paging vs. scrolling on reading online text passages. Usability News, 2003.

[2] L. Baptiste. Perfect PHP pagination. sitepoint. `http://www.sitepoint.com/print/perfect-php-pagination/` (Apr 2009).

[3] M. Bernard, R. Baker, B. Chaparro, and M. Fernandez. Paging vs. scrolling: Examining ways to present search results. In *Proc. of the Human Factors and Ergonomics Society 46th Annual Meeting*, 2002.

[4] T. W. Bickmore and B. N. Schilit. Digestor: Device-independent access to the World Wide Web. In *Selected papers from the sixth international conference on World Wide Web*, pages 1075–1082. Elsevier Science Publishers Ltd., 1997.

[5] A. Blixt. Tabbed navigation using CSS. web. `http://tutorials.mezane.org/tabbed-navigation-using-css/` (Apr 2009).

[6] M. Butler, F. Giannetti, R. Gimson, and T. Wiley. Device independence and the web. *IEEE Internet Computing*, 6(5):81–86, Sep/Oct 2002.

[7] O. Mikheev. Ajax programming with Struts 2. JavaWorld. `http://www.javaworld.com/javaworld/jw-08-2007/jw-08-ajaxtables.html` (Apr 2009).

[8] I. Pepelnjak. Automate the pagination of your web pages. informIT, Feb 2007. `http://www.informit.com/articles/article.aspx?p=691505` (Apr 2009).

[9] A. Peytchev, M. P. Couper, S. E. McCabe, and S. D. Crawford. Web survey design paging versus scrolling. *Public Opinion Quarterly*, 70(4):596–607, 2006.

[10] W3C. CSS Template Layout Module. Web. `http://www.w3.org/TR/2009/WD-css3-layout-20090402/` (Apr 2009).

[11] W3C. CSS3 Module: Paged Media. web. `http://www.w3.org/TR/2006/WD-css3-page-20061010/` (Apr 2009).

[12] YesLogic Pty Ltd. Prince XML. Web, 2008. `http://www.princexml.com/` (Nov 2008).