Document Visualization on Small Displays*

Ka Kit Hoi, Dik Lun Lee, and Jianliang Xu**

Hong Kong University of Science and Technology, Clear Water Bay, HK
dlee@cs.ust.hk

Abstract. Limitation in display size and resolution on mobile devices is one of the main obstacles for wide-spread use of web applications in a wireless environment. Web pages are often too large for a PDA (Personal Digital Assistant) screen to present. The same problem exists for devices with low resolution such as WebTV. Manual reconstruction of web pages for these devices would ease the problem; however, the large variation of display capabilities will greatly increase the burden of web page designers since they have to customize a web page for each possible display device. In this paper, we propose a document segmentation and presentation system. The system automatically divides a web document into a number of logical segments based on the screen size and the structure and content of the document. Additional information such as overviews and summaries is also extracted to facilitate navigation. The system presents the segments and structural information of a web document to make full use of the screen for information finding.

1 Introduction

Mobile computing has drawn much attention in these few years due to the technology advances. However, viewing documents on a mobile device is not an easy task because of the limited display size, CPU power, and bandwidth. This is an obstacle for wireless web browsing as there is no automatic optimal transformation of HTML [12] documents designed for display on large screens to small PDA screens.

The same problem appears in devices for web access (or web-like access) on television. Although the size of TVs generally is larger than most of the ordinary computer monitors, the relatively low resolution and long viewing distance reduce the amount of information they can display. In this paper, we use the term "small displays" to refer to devices which can only display a small amount of information for comfortable user viewing, either because the screens are physically small or have very low resolution.

Small displays increase the burden of web page authors and application designers. They need to consider all the possibilities of the display ability of user's devices when the web pages are created. One of the solutions to the problem is to use a scroll bar to navigate a web page when the page is too large for

* This work was supported in part by grants from the Research Grant Council of Hong Kong (Grant No. HKUST 6079/01E).

** The author is now with Hong Kong Baptist University, Kowloon Tong, HK.

the screen. Obviously, this approach requires many user interactions before the desired information can be reached. Another approach is to break down the web page into small segments in a top-down left-right order and display them one by one. However, users would still have difficulties in locating relevant information if no prior knowledge of the web page is known.

In this paper, an automatic document segmentation and presentation system (DSPS) is presented to solve the problem. The system has three primary functions. Firstly, it automatically divides a web document into different logical segments based on the display size of the devices, and the hierarchical structure and content of the documents. Secondly, it extracts the summary and overview information from the logical segments to help users locate relevant information. Thirdly, an interface for clear and user-friendly presentation of the segments is created for rapid access to the desired information.

The rest of the paper is organized as follows: Section 2 discusses previous work which was done related to the problem; Section 3 describes the algorithms used in the DSPS system; Sections 4 and 5 present the description of the DSPS system and the case studies; lastly, the conclusion is presented in Section 6.

2 Previous Work

2.1 Existing Solutions

The most obvious solution to present information on small display devices is to tailor it for displaying on these devices. There are several information providers who manually create documents for specific mobile devices so that users can download them into their devices [4, 9, 10]. Portability is the major problem for tailor-made information. The same design of a document is difficult to convert from one device to the others. Different formatted versions of the same document are necessary for different kinds of users.

Besides tailor-made methods, there are several web browsers running on different kinds of hand-held devices and webTV devices [1, 6, 11]. The usual technique of these web browsers in presenting web pages is to fill up the screen line by line. This approach provides quite a nice solution for simple web pages, which do not use tables and frames as formatting tools. When a web page contains many tables and frames, which are heavily used for information formatting, the layout of the web page in those web browsers becomes messy and almost impossible to read. The main problem is that they do not perform any kind of content transformation to make the information suitable for display on small screens.

2.2 Studies on Small Display and User Interaction

A number of studies on the effect of reading and understanding of using small displays rather than ordinary displays were carried out. The studies on the effect on reading and comprehension in a small display environment by Duchnicky and Kolers [3], Dillon et al [2], and Shneiderman [14], concluded that a small display

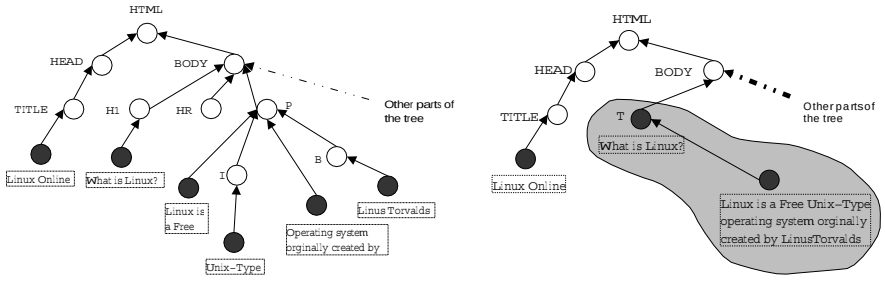
did not dramatically impact the reading speed or the comprehension level of the readers. However, heavy interaction like scrolling would annoy many users. Users had to take a lot more time and key strokes to get their desired information.

An experiment on the impact of small displays which was carried out at the Interaction Design Center, Middlesex University, UK by Matt, Gary, Norliza and Kevin [5] revealed an interesting behavior of users. Twenty computer science staff and student volunteers were asked to view a web site with a common design that is similar to many commercial web sites. In conclusion, large screen users showed a greater tendency to follow navigation paths while small screen users had much shorter navigation paths.

Such finding illustrated some principles in designing or creating web pages for small display devices. Direct access to the information items and search functions play very important roles. The layout of the information should reduce navigation operations like scrolling. Key information of the overall web sites should be presented before the detailed contents of different information items are shown.

2.3 Structured Data Tree, Content Tree and Logical Tree

To process HTML documents, a document model is needed in DSPS. Lim and Ng [7, 8] proposed two types of tree representations of HTML documents, the Structured Data Tree (SDT) and the Content Tree (CT), to represent the hierarchical information of the data contents of the HTML documents. SDT is one kind of parse trees for an HTML document based on the HTML grammar. Fig. 1(a) shows an example of SDT. A CT captures the hierarchical relationships among the contents of an HTML document. Fig. 1(b) shows the CT which is constructed from the SDT in Fig. 1(a).



(a) (b)

Fig. 1. (a) Structured Data Tree (SDT); (b) Content Tree (CT)

Shinagawa and Kitagawa also proposed a structure called Logical Tree which was similar to CT [13]. The construction of the Logical Tree has used context-free grammar which is, in principle, the same as the push down automata approach, which has been proposed by Lim and Ng.

However, there is a serious limitation in both Lim and Ng's algorithm and Shinagawa and Kitagawa's algorithm when they are applied to real-world web

pages. Their algorithms only consider some HTML tags like $H1$, $TITLE$ and $BODY$. Tables and frames, which are heavily used in formatting web pages, are excluded.

3 Converting HTML Document to Content Tree

An HTML document is semi-structured in nature since the high-level structural meaning of the document itself is not explicitly expressed. Also, the HTML grammar provides a high flexibility for controlling the appearance of an HTML document. Such flexibility complicates the process of obtaining useful structural meaning for document processing. In order to obtain high-level structural meaning from an HTML document, this section proposes a series of algorithms and heuristic methods to extract the document structure (expressed in a Content Tree) from an HTML document.

3.1 Overview

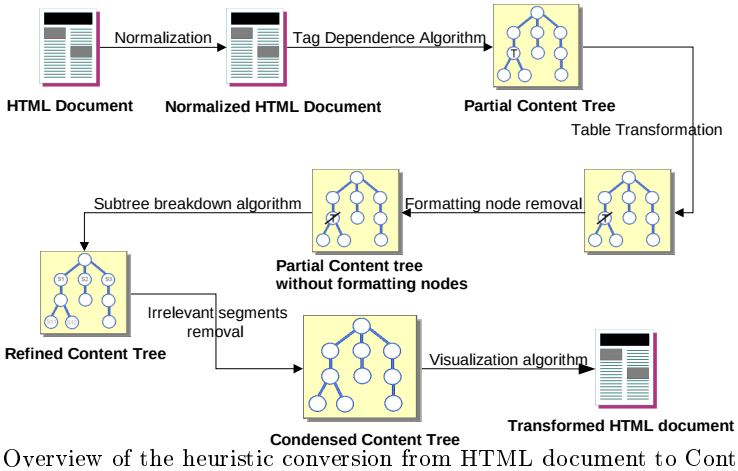


Fig. 2. Overview of the heuristic conversion from HTML document to Content Tree

The objective of all the algorithms that are going to be discussed is to transform an HTML document D into a structured document D_s which can be represented by a CT, CT_{D_s}. The CT_{D_s} should be able to facilitate the visualization of D on a small display screen $Scr(w, l)$. which can only display l lines and each line consists of up to w characters.

An HTML document is first converted and normalized into an intermediate Content Tree by an HTML parser. The intermediate CT is then converted into an initial CT using the *Extended Linear Dependence* algorithm. Then the table refinement algorithm transforms table structures in the HTML document into different subtrees. The *Segment Breakdown* algorithm further refines the CT structure by analyzing the formatting style of each node. Some irrelevant subtree segments are removed from the CT by the *Irrelevant Segment Removal* algorithm. The resulting CT is then transformed into a form that can be used

for visualization on the screen. Fig. 2 shows the overview of the sequence of algorithms used to approximate an HTML document into a CT.

3.2 Classification of HTML Tags

The HTML tags are classified into three types, structural tags, block tags and formatting tags in the algorithms.

- **Structural Tags**: Structural tags contain different structural meanings in the HTML Specification. Among these tags, we can assign the order of dependence. For example, `H1, H2, H3 ...` are structural tags. Based on their semantic meanings, linear dependence order can be assigned to them. (e.g., `H3` is dependent on `H2`, and `H2` is dependent on `H1`).
- **Block Tags**: Block tags should be treated as individual entities. For example, `TABLE`, `TR` and `TD` is a set of tags in which the whole entity is defined between the start and the end tag `TABLE`. The set of block tags can further be classified into block begin tags and block element tags. Block begin tags are the tags that indicate the beginning of the block entity construct. `TABLE` is an example of block begin tag. Block element tags are the building elements of the block entity. `TR` and `TD` are examples of block element tags.
- **Formatting Tags**: Formatting tags used in formatting the text rather than indicating the structural meaning. These tags can be considered as an attribute of that piece of text rather than a node in the CT. These kinds of tags provide heuristic information in the heuristic transformation process described in section 3.6.

Table 1 shows the classification of tags. HTML tags which are not included in the table are discarded in the DSPS to simplify implementation.

Tag Type	HTML Tags
Structural Tag	`Title, H1, H2, H3, H4, H5, H6, P, BR`
Block Begin Tag	`TABLE, UL, OL`
Block Element Tag	`TH, TR, TD, LI`
Formatting Tag	`TT, I, B, U, BIG, SMALL, EM, STRONG, FONT, A`

Table 1. Classification of HTML tags

3.3 Normalization of HTML Document

In order to simplify the algorithms and avoid handling too many special cases, an HTML document needs to be preprocessed and normalized into a simple and clean form.

According to the HTML grammar, formatting tags have very little restrictions on where they can appear. They can enclose a whole table or exist within the table cells. In order to reduce the complexity of the structural analysis, every HTML document should be normalized such that for every path from the root node of the intermediate CT to any leaf node, no formatting tag appears before any structural or block tag on that path.

To accomplish the above condition, all the formatting tags should be shifted down within the intermediate CT. The *Push-down Formatting Tags* algorithm traverses from the root of the intermediate CT to all the leaf nodes. When a formatting tag is encountered, the tag is saved in a stack and then removed from the intermediate CT. All the saved formatting tags will be released when a text leaf node is reached by the algorithms.

The operation does not affect the original structure of an HTML document. The algorithm just deletes a formatting tag and duplicates it to each element of an HTML structure if the formatting tag encloses the structure.

3.4 Extended Linear Dependence Algorithm

The *Extended Linear Dependence* algorithm is based on the construction of CT proposed by Lim and Ng [7, 8]. The original construction of CT is based on the tag object dependence. The linear dependence defines a sequence of the tag objects (`H1 > H2 > H3 ..... `) which indicates the conceptual, structural and scope meaning of the tags. Any access path from the root node to any leaf node of the CT should follow the linear dependence.

Structural Tags		Formatting Tag
`TITLE, H1, H2, H3, H4, H5, H6`	`P`	`(TT, I, B, BIG ,SMALL,`
		`EM, STRONG, FONT, A)`
	`BR`	
	`UL, LI`	(all the formatting tags
	`OL, LI`	are of the same dependence value)
Lower dependence value		Higher dependence value

Table 2. Dependence order of HTML tags

In order to cope with the problem, we extend Lim and Ng's algorithm to create an initial CT for a real-world web page. The main element of the extension is to use different strategies for different types of tags. Different algorithms are applied to the structural, block and formatting tags. First, let us define the linear dependence values of the structural and formatting tags. Table 2 shows the linear dependence among the HTML tags.

The algorithm of creating an initial CT from an intermediate CT, CT_{SDT}, is defined as follows. Define a Block Elements Restricted In-order Traversal on a node n as the in-order traversal that begins at node n but excludes the subtrees rooted at a block begin tag. The block begin tags are included in the traversal.

Define Linear Dependence Transformation Function $ldf(\text{node}n_r)$ as follows.

1. Create a list of nodes $L_{CT_{SDT}}$ with Block Elements Restricted In-order Traversal on node n_r.
2. Create a content subtree root node $sroot = (n_r.tag, \text{“”})$.
3. Create an empty stack BS.
4. Set *currnode* to *sroot*.
5. For each node cn_i in $L_{CT_{SDT}}$
   (a) create a node $n_i(cn_i.tag, \text{“”})$
   (b) if cn_i is a leaf node, set $n_i.text = cn_i.body$

(c) if $cn_i.tag$ is a block begin tag, attach the node n_i to $currnode$ as a child, push the (cn_i, n_i) pair into BS

(d) if $cn_i.tag$ is a structural tag or formatting tag and the dependence value of $cn_i.tag$ is larger than the dependence value of $currnode.tag$, i.e. $cn_i.tag$ depends on $currnode.tag$, attach n_i to $currnode$ as a child, set $currnode = n_i$

(e) otherwise, set $currnode$ = the parent node of $currnode$, repeat step d

6. For each pair (cn, n) in BS
   (a) attach the return subtree of $\mathtt{ldf}(cn)$ to n
7. Return the subtree rooted at $sroot$.

The transformation of the intermediate CT to an initial CT is done by applying the `ldf` function on the root node of the intermediate CT. The node returned by the algorithm is the root node of the initial CT. Fig. 3 shows an example of an initial CT.

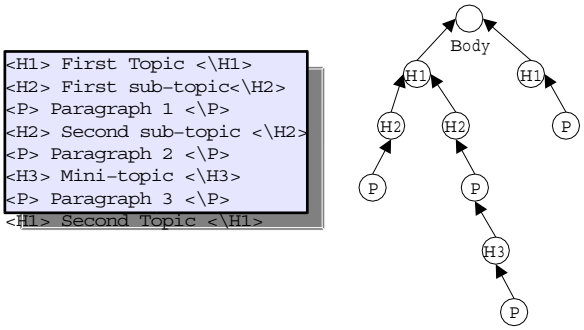


Fig. 3. Example of Extended Linear Dependence algorithm

3.5 Table Analysis

Table is heavily used in formatting web pages due to the fact that normal HTML tags cannot provide high formatting power. However, this makes the extraction of the actual logical structure from a web page much more complicated. This subsection proposes a heuristic approach to analyze the table structures in the HTML documents. In the following, we first describe several assumptions.

We assume that tables contain some textual content. Tables containing only figures can be identified at the parsing stage using some heuristic measures such as cell size. Thus, each table can be transformed into a single node in the construction of the initial CT. We also assume that there is no column or row span in the table. If column and row spans are used, the table is transformed into a normal table by splitting the spanning columns or rows and duplicating the content of the spanning cell accordingly.

Another assumption for the table analysis is that the text segments belong to the same level of the logical hierarchy do not span across different columns. For example, if all the text segments belong to a section, all of them will be located within one column. As such, the logical hierarchy will not break when the table analysis is separating two adjacent columns. The assumption is valid for most

of the web pages because HTML pages do not have limits on their length. It is not necessary to split a logical paragraph into two columns.

Based on the above assumptions, we can transform a table as follows. Let $T(m, n)$ be a table of m columns by n rows, and each cell of the table represented as $c(i, j)$ where $0 < i < m$ and $0 < j < n$. Let $S_{(i,j)}$ be the sectional subtree on $c(i, j)$ produced by the *Extended Linear Dependence* algorithm. An initial CT of $T(m, n)$ is constructed by the following *Table Transformation* algorithm:

1. Create a root node *root*(`TABLE`, "").
2. Set *currnode* to *root*.
3. For each column i of $T(m, n)$
   (a) create a node *cell*(`C`, "")
   (b) set *cell* to *root* and set *currnode* to *cell*
   (c) for each row j of column i
      i. create a node *para*(`P`, "")
      ii. append the root node of $S_{(i,j)}$ to *para* node
      iii. append the node *para* to *currnode*

The algorithm classifies each column of the table structure as a separated logical unit. It also assumes that cells in the same column are highly related but not related to the cells in other columns.

3.6 Heuristic Transformation

After performing the *Extended Linear Dependence* algorithm and *Table Transformation* algorithm, the resulting CT captures most of the structural information provided by the structural tags and table structures. However, not every web page authors or web page authoring tools use structural tags or table structures to indicate the structural information of the web pages.

If the HTML fragment does not contain structural tags or tables, heuristic can be applied to fill up the gap. In the following section, the use of formatting styles of the web pages to extract hidden structural meanings from the web pages is discussed. The idea behind the heuristic is that if a small piece of text contains a lot of formatting, it is more important and structurally meaningful than a plain text is.

Transformation of Formatting Nodes A node is defined as a formatting node if the tag of the node is a formatting tag. Formatting nodes complicate the structural analysis, since a heavily formatted text segment would become a large subtree. In order to facilitate the heuristic breakdown algorithm, this kind of nodes need to be removed. We introduce the concept of formatting value which can simplify a complicated subtree structure into a single text node together with a quantified measure of the formatting style.

Different formatting tags have different degrees of emphasizing effects. For example, `B` has a stronger emphasizing effect than `I` and `SMALL`. Each formatting tag is assigned with a value, a higher value means a higher degree of emphasizing effect. Table 3 shows the formatting value of each formatting tag in our DSPS implementation. The value is derived by estimating the emphasizing effect of each formatting tag.

Formatting Tag	Formatting Value	Formatting Tag	Formatting Value
STRONG	3	I	1
BOLD	2	EM	1
BIG	2	U	1
FONT with size inc.	3	TT	0
FONT with size dec.	-1	SMALL	-1
A	2		

Table 3. Formatting value of formatting tags

The *Formatting Nodes Removal* algorithm removes all the formatting nodes by using a numeric value to replace a collection of formatting nodes. The numeric value reflects how heavy the text is formatted. The numeric value is calculated by summing up all the formatting values of the formatting tags applied to the text. Each nodes in the CT will be assigned a formatting value after the *Formatting Nodes Removal* algorithm. Fig. 4 shows an example of the *Formatting Nodes Removal* algorithm.

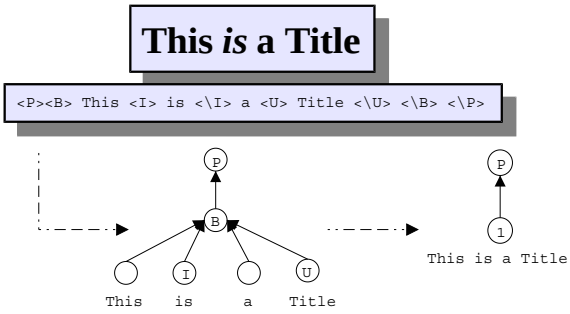


Fig. 4. Example of Formatting Nodes Removal algorithm

Segment Breakdown Algorithm The basic idea of the *Segment Breakdown* algorithm is to segment an HTML segment by identifying titles within the segment and using them as the cutting points to create sub-segments. The *Segment Breakdown* algorithm can be applied to each HTML segment until the size of a sub-segment is small enough for display.

The *Segment Breakdown* algorithm identifies the highly formatted text nodes and promote them to a higher hierarchical level of the CT. Less formatted texts then follow the highly formatted texts as dependent nodes. The algorithm can break a flat tree structure into a hierarchical one by analyzing the formatting styles. This operation is essential in DSPS for visualizing the document on small display devices as the document fragment represented by the flat tree structure can be replaced by a number of logically separated segments. Also, an overview, which consists of highly formatted texts, of these segments is also generated.

3.7 Content Analysis

In order to provide more relevant and concise information to the users, analysis on the textual content of the CT is needed.

Irrelevant Segment Removal Algorithm It is common that there are some irrelevant segments in HTML documents, especially for commercial web

pages. Irrelevant segments can be eliminated so that the HTML document can be visualized on small display device effectively. In the design of the algorithm, two assumptions are made for irrelevant segments: 1) irrelevant segments are usually small in size; 2) irrelevant segments have no or few keywords in common with the relevant segments within the same logical level.

For a node n in a CT, define Seg_n as the text of the segment represented by the subtree rooted at n. Let $SIM(t1, t2)$ be the similarity scores of text $t1$ and $t2$. The algorithm for calculating the relevance scores RS_{n_i} of a node n_i is given:

1. Let $n_1, n_2 ... n_{i-1}, n_{i+1} ... n_m$ be the sibling of n_i.
2. Set text $t1 = \cup(Seg_{n_1}, Seg_{n_2} ... Seg_{n_{i-1}}, Seg_{n_{i+1}} ... Seg_{n_m})$.
3. Set text $t2 = Seg_{n_i}$.
4. $RS_{n_i} = SIM(t1, t2)$.

Subtrees with relevance scores which are lower than threshold tr are considered irrelevant and can be purged from the original CT to provide more precise information. In practice, the relevance scores of all sibling subtrees at the same level of the logical hierarchy are normalized to the range of 0 to 1. Thus a single threshold value can be used in different levels of logical hierarchy.

The conceptual meaning of the algorithm is based on the fact that a segment should be irrelevant if it is not similar to the rest of the text that is within the same level of the logical hierarchy.

Similarity Measure The Boolean vector space model is used to calculate the similarity scores in our implementation of the DSPS.

Let the keyword space be $(w_1, w_2, ... w_n)$. Let $v_t(b_1, b_2, ... b_n)$ be the bit vector of text t, such that

$$b_i = \begin{cases} 1 & \text{if } t \text{ contains } w_i \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Given two document vectors v_1, v_2, the scores $SIM(v_1, v_2) = \frac{|v_1 \cdot v_2|}{\sqrt{|v_1| + |v_2|}}$. The reason for using the size of the vector as the divisor is to minimize effects of the long paragraph, especially in intra-document analysis. This is because most of the entities in a document are titles, subtitles and short paragraphs which will be unfairly treated if normalization based on segment size is not used.

Special Case The *Irrelevant Segment Removal* algorithm has an interesting behavior when we apply it to subtrees with a very small number of siblings. The scores for all sibling would be very low if they have too few or even no common keywords. For example, if there are two children of a node and they have no common keywords, the scores of the first sibling and the second one are both zero. The algorithm cannot conclude which child is irrelevant. Thus the DSPS will not remove any children if all the children have scores lower than a minimum threshold.

4 Document Segmentation and Presentation System

4.1 System Overview

The Document Segmentation and Presentation System was developed using the Java2 Platform [15]. The DSPS is a Java application which contains four internal windows within a main window. These four windows are the original HTML document, the complete CT, the CT with irrelevant segments removed and the card view of the transformed HTML documents. Both CTs are presented using the Swing tree component of the Java 2 Swing package.

The Java2 Platform contains a robust HTML parser and supports both the Extended Markup Language (XML) and the Document Object Model (DOM). As most of the operations in the DSPS interact with the tree structures like SDT and CT, the XML and DOM support is essential. There is a clear mapping from a tree structure to an XML document or DOM structure.

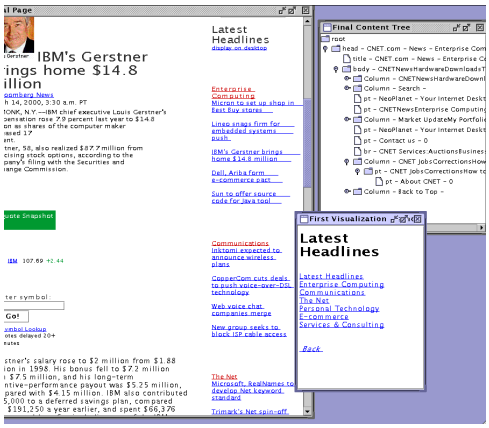


Fig. 5. Document Segmentation and Presentation System screen shot

Internally, DSPS consists of two main sub-systems, the HTML document transformation sub-system and the document visualization sub-system. DSPS takes two inputs, the URL of the target HTML document which is passed in as a command line argument, and the information of the display which is stored in a property file. A transformed HTML document is generated after a series of operations.

4.2 HTML Document Transformation

The HTML document transformation sub-system is the critical part of the DSPS. It takes an HTML document as input and transforms the unstructured HTML document into a meaningful and structural data which is represented as a CT structure, following three steps:

1. **HTML Document Parsing:** DSPS first converts the HTML document into an intermediate CT using the Hot-Java HTML parser, which was built

inside the Java 2 Platform, by implementing the call back functions. The call back functions are called when appropriate tokens are encountered. The Hot-Java HTML parser provides error handling and recovery on some common HTML errors like inserting missing end tags and rearranging wrongly ordered HTML tags. DOM has been used as the underlying data structure to represent the CT structure.

2. **Content Tree Construction:** The normalized intermediate CT is transformed into an initial CT using the *Extended Linear Dependence* algorithm. All the tags information including the type and the dependence information have been stored in different property files. The initial CT structure has also been represented by DOM.
3. **Content Tree Refinement:** The *Formatting Tags Removal* algorithm has been applied to the initial CT. The formatting value of each node in the CT has been stored in the attribute of each node. All the formatting values of the formatting tags have been stored in a property file.

4.3 Document Visualization

For a specific device, the CT of the HTML document needs to be customized in order to obtain an adaptive visualization effect. The DSPS needs to transform the current CT first to provide suitable views of the document for the device.

1. **Content Tree Customization:** The *Segment Breakdown* algorithm breaks down the flat structure of the CT into hierarchical structure by incorporating two factors, the formatting values of all the text segments and the size of the display. Therefore, the resulting CT contains subtrees that can fit into the corresponding devices. The *Irrelevant Segment Removal* algorithm removes the irrelevant segments from the CT to provide precise information to the users.
2. **Card-View Presentation:** After the device-specified CT has been obtained, the display engine visualizes the CT using the card-view methodology. The display engine converts the CT into a deck of cards. The content of each card consists of three elements, the text of the node as the header, the list of the texts of all its children and a backward hyperlink. Each text in the list contains a hyperlink to the card of the corresponding child node. The backward hyperlink is used to access the parent of the current card. The first card of the document is the overview which is generated by performing the overview extraction operation on the CT. The depth of the CT to be extracted is adjusted to an optimal level such that the overview can fit into the display of the device. Users can access the information by first looking at the overview of the document and then selecting the header to access different logical sections of the document. The card-view method is suitable for small displays since the size of each card is small but users can obtain the overall picture of the document by accessing the top level overview.

5 Case Studies

This section demonstrates how the DSPS converts some important components of the real-world web pages into different Content Tree structures and thus shows

the strengths and weaknesses of the DSPS. The effect of each algorithm is highlighted and explained for critical parts of the documents.

5.1 Case 1 : Wired.com - Front Page

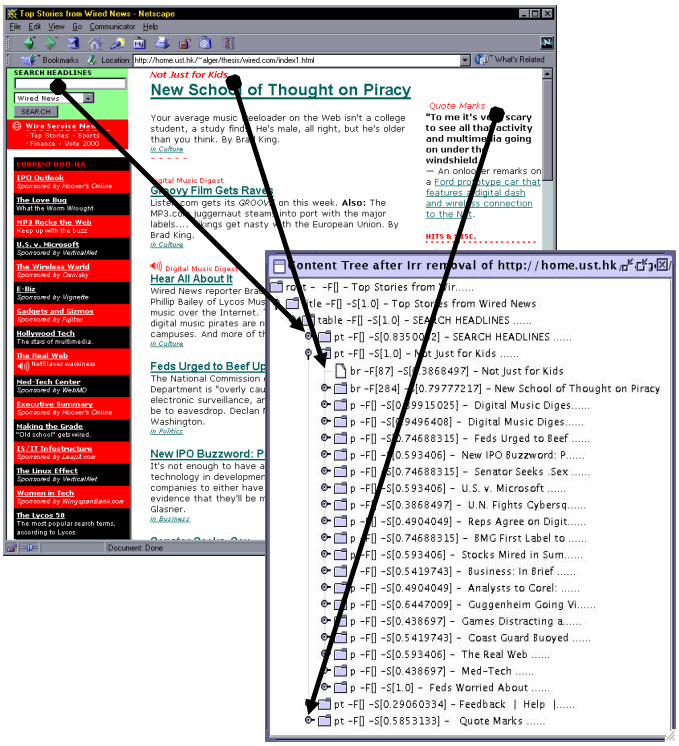


Fig. 6. HTML document of the front page from Wired.com and its CT

Let us look at the front page of Wired.com first. As we see in Fig. 6, the page contains links to other places in the site on both sides of the page while the center part contains the collection of news items.

Regarding to the three main segments in the page, let us define $seg1$ as the left column represented by the subtree which is rooted at "`SEARCH HEADLINES`", $seg2$ as the center column represented by the subtree which is rooted at "`Not Just for Kids`" and $seg3$ as the right column represented by the subtree which is rooted at "`Quite Marks`" (See Fig. 6). $seg1$ and $seg3$ contain lists of links to other places in the web site which could be considered as irrelevant segments. The scores for $seg1$ and $seg3$ are 0.8350 and 0.5853 respectively. If we set the irrelevant threshold to 0.9, $seg2$ which has score of 1.0 will not be purged. $Seg1$ and $seg3$ will be considered irrelevant and removed from the CT. The result of irrelevant segment purging is satisfactory in this case.

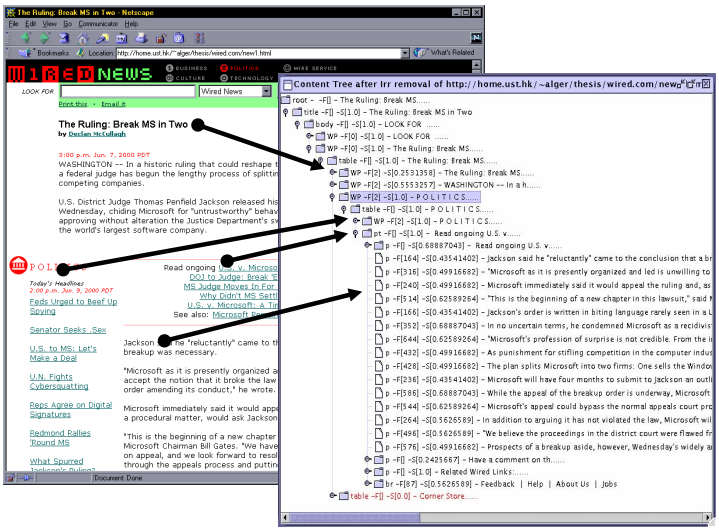


Fig. 7. HTML document of the article from Wired.com and its CT

5.2 Case 2: Wired.com - Article

The news article in Fig. 7 of the Wired.com shows the pitfall of the *Irrelevant Segments Removal* algorithm. Considering two segments in the page, let us define *seg*4 as the left column represented by the subtree which is rooted at "`POLITICS` ⋯" and *seg*5 as the center column represented by the subtree that is rooted at "`Read ongoing US vs` ⋯."

In the CT, there exists only two siblings at the level of *seg*4 and *seg*5 (indicated by the arrow in Fig. 7). The scores for these two segments are the same, i.e., 1.0. It cannot be concluded that which one is more relevant than the other. One way to solve this problem is to use the size of the segment as a heuristic to decide which segment can be purged.

5.3 Case 3 : Linuxdevices.com - Article

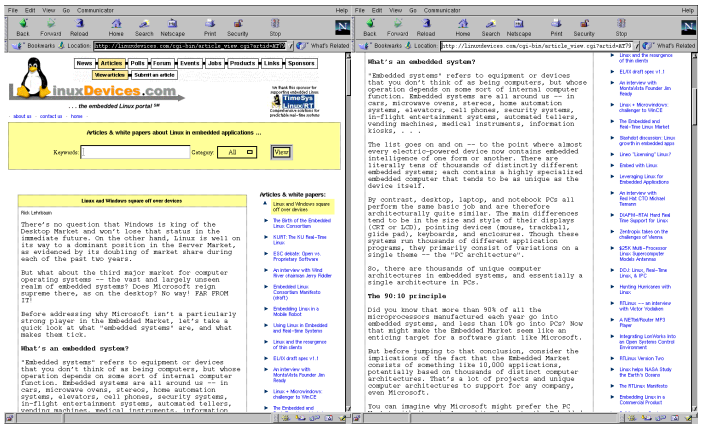


Fig. 8. Article from LinuxDevices.com

We use an article from Linuxdevices.com to illustrate how DPSP processes an HTML document which contains a long passage composed of a collection of paragraphs. Fig. 8 shows the snapshot of the sample article. The formatting style of this article is similar to that of the front page of Wired.com. Banners are on the top of the pages. A two-column table forms the main body of the article. The long passage locates in the first column while a listing of articles locates in the second column.

Unlike the front page of Wired.com, the long passage at the center column is composed of a collection of paragraphs with `BR` as the newline delimiter. The long passage can be viewed as a collection of logical segments and each logical segment is in the form of a header followed by a number of paragraphs. The logical header is highlighted by using formatting tags such as `B` and `FONT`, instead of using structural tags such as `H1` and `H2`.

As the structure of the article is similar to the front page of Wired.com, we focus on the long passage and describe the effect of the heuristic breakdown algorithm. During the first stage of parsing, a flat tree is obtained. Each node is a text segment which is separated by tags `<P>` or `<BR>`. As no structural tag exists to help breaking down the flat tree into a more structural one, Segment Breakdown algorithm plays an important role in solving the problem.

It is obvious that the formatting value of the header in each logical unit is higher than those of the paragraphs. By calculating the formatting value of each node and applying the Segment Breakdown algorithm, the flat tree becomes a more structural one. Each header becomes the root node of its associated paragraphs and all the headers becomes siblings. The resulting CT is shown in Fig. 9.

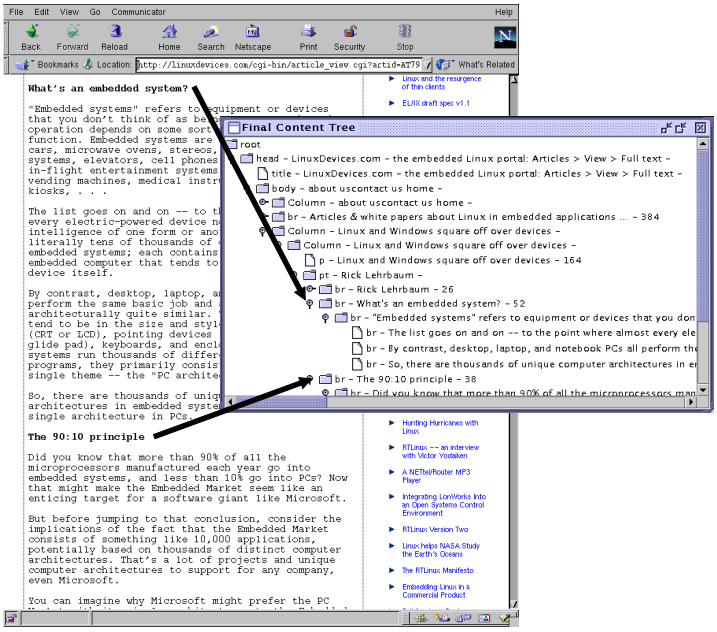


Fig. 9. CT of the article from LinuxDevices.com

6 Conclusions

In this paper, we proposed a document segmentation and presentation system (DSPS) to transform an HTML document into a data structure called a "Content Tree", which represents the logical structure of the HTML document. The Content Tree structure can be used to provide various views that are suitable for display in various small displays. One of interesting ideas of the paper is the introduction of the notion of formatting value, which helps to simplify formatting structures and hence ease structural analysis. The DSPS is targeted for real world commercial HTML documents that contain many screen-oriented formatting styles like heavily nested tables and complicated text formatting structures.

References

1. F. G.-Castano et al. Hierarchical atomic navigation for small display devices. In *Proc. WWW10*, 2001.
2. A. Dillon, J. Richardson, and C. McKnight. The effect of display size and text splitting on reading lengthy text from the screen. *Behaviour and Information Technology*, 9(3):215–227, 1990.
3. R.L. Duchnicky and P.A. Kolers. Readability of the text scrolled on visual display terminals as a function of windows size. *Human Factores*, 25:683–692, 1983.
4. InfoRover. *InfoRover*. Commercial company delivers transformed web contents to Palm Pilot handheld devices. (http://www.inforover.com).
5. Matt Jones, Gary Marsden, Norliza Mohd-Nasir, and Kevin Bonne. Improving web interaction on small displays. Technical report, Interaction Design Centre, School of Computing Science, Middlesex University, 1997.
6. Oku Kazuho. *Palmscape PR 5.0a5*. A free web browser running on Palm Pilot handheld devices. (http://www.muchy.com/review/palmscapeloc.html).
7. Seung Jin Lim and Yiu Kai Ng. Constructing hierarchical information structures of sub-page level html documents. In *Proceedings of the 5th International Conference of Foundations of Data Organization (FODO'98), Kobe, Japan*, pages 66–75, 1998.
8. Seung Jin Lim and Yiu Kai Ng. Extracting structures of html documents. In *Proceedings of the 12th International Conference on Information Networking (ICOIN-12), Tokyo, Japan*, pages 420–426, 1998.
9. Microsoft. *Microsoft WebTV*. Commercial company provides web-based access services through TVs. (http://www.webtv.com).
10. Pocket Info. *Pocket Info*. A public domain initiative in 1998 to provide information for the users of handheld computers, palmtops and organizers. (http://www.pocketinfo.org).
11. ProxiNet. *ProxiWeb*. A web browser running on Palm Pilot handheld devices. (http://www.proxinet.com).
12. D. Raggett. *HTML 3.2 Reference Specification*. World-Wide Web Consortium, 1997. http://www.w3c.org/TR/REC-html32.
13. Norihide Shinagawa and Hiroyuki Kitagawa. Dynamic generation and browsing of virtual www space based on user profiles. In *Proceedings of the Fifth International Computer Science Conference, Hong Kong*, pages 93–108, 1999.
14. B. Shneiderman. User interface design and evaluation for an electronic encyclopedia. *Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems*, pages 207–223, 1987.
15. Sun Microsystem. *Java 2 Platform*, 1998. http://java.sun.com.