DRESS: A Slicing Tree Based Web Representation for Various Display Sizes

Liqun Chen
Xing Xie
Wei-Ying Ma
Hong-Jiang Zhang
Heqin Zhou
Huanqing Feng

2002-11-16

Technical Report
MSR-TR-2002-126

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

DRESS: A Slicing Tree Based Web Representation for Various Display Sizes

Li-Qun Chen*†, Xing Xie‡, Wei-Ying Ma‡, Hong-Jiang Zhang‡, Heqin Zhou†, Huanqing Feng†

† School of Info. Sci. & Tech.
Univ. of Sci. & Tech. of China
Hefei, 230027, China
+86-551-3601510

‡Microsoft Research Asia
No. 49, Zhichun Road
Beijing, 100080, China
+86-10-62617711

lqchen80@mail.ustc.edu.cn
{hqzhou, hqfeng}@ustc.edu.cn

{i-xingx, wyma, hjzhang}@microsoft.com

ABSTRACT

As a great many of new devices with diverse capabilities are making a population boom on the Internet mobile clients, their limited display sizes become a serious obstacle to information accessing. In this paper, we introduce a *Document REpresentation for Scalable Structure (DRESS)* to help information providers to make composite documents, typically Web pages, scalable in both logic and layout structure to support effective information acquisition in heterogeneous environments. Through this novel document representation structure based on binary slicing trees, the document can dynamically adapt its presentation according to display sizes by maximizing the information throughput to users. We discuss the details of this structure with its key attributes. A branch-and-bound algorithm and a capacity ratio based slicing method are proposed to select proper content representation and aesthetic document layouts respectively. Experiments show satisfactory results with high efficiency.

Keywords

Web browsing, adaptive content delivery, slicing tree, layout optimization

1. INTRODUCTION

Recently, a great many of new devices with diverse capabilities, such as Tablet PCs, Pocket PCs, Smartphones and Handheld PCs, are making a population boom on the Internet mobile clients because of their portability and mobility. Since most of the information on Internet today is presented by Web documents, there has been an imperative need for efficient access to the bulk of Web resources on these diverse-form-factor devices. Although these devices are becoming more and more powerful in both numerical computing and data storage, nevertheless, low bandwidth connections and small displays, the two serious obstacles to information accessing, have prevented them becoming more helpful to people's everyday life. With the successful development of scalable video coding, progressive image coding as well as 2.5G and 3G wireless networks, the bandwidth condition is expected to be greatly improved in the near future. At the same time, however, the limitation on display size is more likely to remain unchanged for a certain period of time due to the mobility requirement of these devices and the slow progress of foldable display technology.

Aiming at solving this problem, we introduce a *Document REpresentation for Scalable Structure (DRESS)* based on the combination of techniques rooted in computer aided design and linguistic summarization. When the display area shrinks, some parts of the Web pages will be compressed in terms of linguistic summaries by computational information extraction or manual input from the page author, and then presented, together with other unsummarized parts, adaptively to end users with aesthetic layouts.

The rest of this paper is organized as follows: In Section 2, we discuss the state of art. Section 3 presents the definition of DRESS representation and its main attributes. This is followed by Section 4 introducing the Web page rendering algorithm. Then, in Section 5, we describe in detail the implementation of a DRESS-based Web browser and present experimental results as well. Finally, Section 6 gives our conclusions and future work.

2. RELATED WORK

Many efforts have already addressed the problem of Web browsing on small terminals and various solutions have been proposed including some commercial products. Current approaches can be divided into two directions: the first one is to transform existing Web pages such as [2][3][5][6][12][15][16][18][20], while the other attempts to introduce new formats and mechanisms [1][4][13][14] which make Web pages themselves scalable to different display sizes.

Among the first trend, there are also two different approaches for transforming Web pages. The first one, which originated from the user interface community, only changes the presentation of page contents without any structure modification. For example, the most straightforward approach is eliminating the annoying horizontal scrolling requirement, i.e. present all the contents into a single narrow column, such as [16]. Fast and simple though it is in implementation, this method greatly increases the page height and forces the user to scroll up and down excessively. Other UI-based approaches try to use thumbnails or keywords as well as zooming techniques to aid browsing, such as [3][18]. However, this kind of aid tends to work only on the pages that user is very familiar with, because thumbnails are often scaled down too much to give much information besides a rough overview. The other way to do Web page transformation is to retrieve the semantic structure from original contents and rewrite the page according to user's context. The basic idea is to partition the Web page into a

* This work was performed when the first author was a visiting student at Microsoft Research Asia.

set of sub-pages and generate a Table-of-Content with/without hierarchy as the index page, such as [2][5]. Our previous work [6][12][20] belongs to this category. A promising direction is to combine these two techniques together to provide a better user experience, for example, [15] uses the whole page thumbnail as the Table-of-Content.

Although many efforts have been put on automatic extraction of document architectures, it is still hard for computers to fully understand the semantic structure of Web pages. What's more, most of them did not address the layout problem which is very essential because the representation of contents directly affects user's perception. Therefore, instead of transforming existing Web pages, researchers began to consider leveraging extra hints from author's direct help in the designing phrase or some external annotations from third parties. A straightforward approach comes along with the XML and XSLT, which allows the semantic structure to be separated with presentations, such as [13].

However, writing a separate style sheet for each type of devices could be very labor intensive partly due to the great variability of some application parameters like window sizes. A more scalable approach for defining Web presentations should be studied. In [14], instead of using multiple presentations, content adaptation is performed based on annotations of adaptation hints, but the layout problem of adapted results has not been addressed either. [1][4] proposed a CSS compatible mechanism which allowed Web page designers or editors to designate layout constraints explicitly in mathematic equalities or inequalities, and then turned the display problem into constraint solving. Despite constraints for interactive graphical applications has been researched since the early eighties of last century, there are still a number of unsolved problems with constraint-based layout. For instance, it is difficult to specify proper mathematical formulas for different layouts. In addition, the constraint solving procedure is computationally expensive for client browsers. There are also some interesting solutions for specific problems such as [10][11] for Web newspaper layout and [9] for optimizing picture placement in a Web page.

We focus on the problem of defining a scalable and efficient Web document representation structure that is adaptive to various display sizes. Here the display size does not need to be as same as the screen size. Actually, in practice each user can designate a preferred target display size different from the screen size. For instance, a Pocket PC user may specify the limitation of his display to a size of two screens with vertical scrolling only. He/she can even use a rotated display which may fit current Web pages better.

As we notice, few of current approaches to Web layout adaptation considered the priorities of different content blocks in a page. What's more, none of them let the author control the final page layout conveniently. That is to say, the final presentation is usually unpredictable during the designing phase. In this paper, we propose to use binary slicing trees, a data structure widely used in computer aided design community [8], to represent a page template that can be controlled by the author. Based on this representation, we formulate the Web layout problem as a variance of 0-1 knapsack problem, which can be efficiently solved by a branch-and-bound algorithm. A capacity ratio based slicing method is then applied to select an aesthetic layout.

3. DOCUMENT REPRESENTATION FOR SCALABLE STRUCTURE

In this section, we first give a formal definition of DRESS, the basis of our Web layout adaptation algorithm, and then describe this structure in detail.

Definition 1: The DRESS for a Web page is a binary slicing tree with N leaf nodes. Each inner node is labeled with either v or h denoting vertical or horizontal split, and each leaf node is an information block defined as follows:

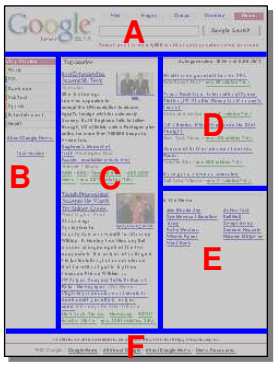
$$B_i = (IMP_i, MPS_i, MPH_i, MPW_i, ADJ_i, ALT_i) \quad (1)$$

where $i = 1, 2, ..., N$,

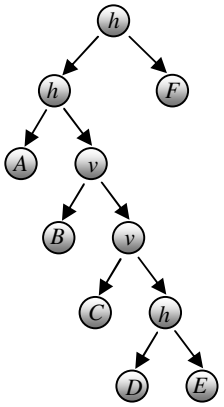
B_i,	the i^{th} information block in the Web page
IMP_i,	importance value of B_i
MPS_i,	minimal perceptible size of B_i
MPH_i,	minimal perceptible height of B_i
MPW_i,	minimal perceptible width of B_i
ADJ_i,	whether the aspect ratio of B_i is adjustable
ALT_i,	alternative of B_i

3.1 Slicing Tree Structure

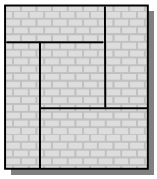
As shown in Definition 1, DRESS is a slicing tree with each leaf node as an information block in the Web page. The label on each inner node determines how the display area is recursively subdivided into sub-rectangles by slicing vertically (v) or horizontally (h). An information block will be placed in the sub-rectangle held by the leaf node.



(a)



(b)



(c)

Figure 1. (a) An example Web page. (b) The corresponding slicing tree. (c) A Web layout that cannot be represented by the slicing tree structure.

An example Web page and its corresponding slicing tree are shown in Figure 1 (a) and (b) respectively. It comes from Google news (http://news.google.com/) and will also be used in the experiment section. Our definition of slicing tree template does

not cover where to split, i.e. the ratio of each split, which is often referred to as the slicing number. In our approach, all the slicing numbers will be adjusted adaptively to the display size. It should also be noted that the slicing tree structure could not represent all kinds of layout. A contrary example is shown in Figure 1(c), however most Web pages in the real world can be represented by slicing trees.

Different from traditional CAD approaches, we assume this slicing tree structure remains constant during the whole adaptation process. This is reasonable since Web authors usually do not want their contents to be randomly shuffled after adaptation. Therefore, only the slicing numbers need to be decided during page adaptation. Based on this assumption, we design an efficient top-down layout algorithm which will be discussed in Section 4.3.2.

Another benefit of using slicing trees is that it usually reflects both the intended logical structure and layout structure of the author. Considering the process of designing a Web page, especially a large one, the author usually follows a top-down flow. First, he/she divides the whole page into several large blocks logically independent, such as header, footer, main topics in body, and side bars. And then, the author fills each block with desired contents as well as some decorations or separators to further divide the block into sub-blocks. Therefore, each block becomes a basic unit to deliver information and attract user attention, as called information block (B_i) here. We assign six attributes to each information block: *importance value (IMP)*, *minimal perceptible size (MPS)*, *minimal perceptible height (MPH)*, *minimal perceptible width (MPW)*, *adjustability (ADJ)*, and *alternative (ALT)*. The following parts will introduce each of them in detail. Though here the slicing tree template is supposed to be written by the author in advance, it may be obtained automatically by certain layout detection approaches, such as the object projection algorithm proposed in [12].

3.2 Importance Value

Since different blocks carry different amount of information and have different functions, they are of different importance. Thus, we introduce *importance value (IMP)*, a quantified value of author's subjective evaluation on an information block, as an indicator of the weight of each block in contribution to the whole page information. This value is used when choosing the less important blocks for summarization under small displays. For simplicity, we normalize all the weights of information blocks in a single page.

Although this value is primarily introduced for the page author to discriminate different contents, however, user preference may also make its contribution to the importance value when considering personalization [19]. For example, we may adjust the *importance value* to user's current focus like the mouse position. We can also employ some automatic approaches [6] to estimate the importance value from block function, position, and size. For instance, we can set the footer block to be less important in a news Web page.

3.3 Minimal Perceptible Size/Height/Width

These attributes are mainly used in the layout optimization. As to document rendering, we can apply many techniques to accommodate diverse target area sizes, such as zooming and scaling, wrapping, font size reduction, or margin cropping. Obviously, the information delivery of a block is significantly relying on its area of presentation. If an information block is scaled down too much, it may not be perceptible enough to let users catch the information that authors intend to deliver. Therefore, we introduce *minimal perceptible size (MPS)*, *minimal perceptible height (MPH)*, and *minimal perceptible width (MPW)*, to denote the minimal allowable spatial area, height, and width of an information block, respectively. They are used as thresholds to determine whether an information block should be shrunken or summarized when rendering the adapted view.

For instance, consider an information block of a short news story whose original region size is 30,000 pixels. The author or publisher may define its *MPS* to be an area of half scaled, i.e. 7,500 pixels, which is assumed to be the smallest resolution to keep the text still readable. He/she can also set the *MPH* to the height of a 9-point character so that the text can be displayed correctly.

Strictly speaking, *MPS*, *MPH* and *MPW* are dependent on the usage context, such as the user's eyesight and his distance from the screen. For example, if a Web page is shown on a TV set with set-top box installed, they should be tuned bigger, since users usually operate it via a remote control. *MPS* also brings us an implication on how much content it is in an information block. This is proved to be very useful in our layout algorithm. *MPS*, *MPH* and *MPW* can be automatically calculated according to the total number of characters, height of one character, and the longest word within a text paragraph, respectively.

3.4 Adjustability

Adjustability (ADJ) denotes whether the aspect ratio of an information block is adjustable. For example, if the content block is a pure text block or a mixture of images and texts, e.g. a news paragraph, it can be wrapped and adapted to fit into the aspect ratio of final display region. However, when the information block is a table like navigation bar, or a large image, the aspect ratio is usually fixed. In this case, the value of *ADJ* should be set to false and we fix its aspect ratio at *MPW/MPH*. This attribute is used in the content accommodation step described in Section 4.3.3. If not specified, *ADJ* can be decided by analyzing HTML tags.

3.5 Alternative

As regards to those information blocks of less importance such as decorations or advertisements, it is desirable to summarize them in order to save display space for more important blocks. Also when dealing with a block of large *MPS* which cannot be displayed without excessive shrinking due to the limited display size, a summary with a link to the original contents is more preferable. Instead of deleting contents or showing imperceptible adapted version, alternative lets users see the whole in parts and give a much better solution to preserve contents, save display space, and aid user navigation as well.

An alternative is usually a short text string that briefly describes the original content block. Its function is similar to the *ALT* attribute of *IMG* tag in HTML. It should not be too big in size due to its function requirements. The summary can be obtained by computational information extraction or manual input from the author. In current DRESS representation, only leaf node can have *ALT* attribute, since summarizing a sub-tree will change the template defined by the author and also increase the depth of page hierarchy. We plan to investigate the possibility of sub-tree summarization in future work.

4. DRESS BASED DOCUMENT LAYOUT ADAPTATION

Based on the previously described DRESS, the problem of Web document layout adaptation can be better handled to accommodate both author intention and user context. In the following, we will discuss the concept of information fidelity and an algorithm to find the optimal Web document layout under various constraints on display size.

4.1 Information Fidelity

Information fidelity introduced here is the perceptual 'look and feel' of a modified version of content object, a subjective comparison with the original version. The value of information fidelity is confined between 0 (lowest, all information lost) and 1 (highest, all information kept just as original). Information fidelity gives a quantitative evaluation of content representation. Hence, the optimal solution is to maximize the information fidelity which is delivered to the end user via an adapted representation. The information fidelity of an individual adapted block is decided by various parameters such as spatial region of display, content reduction of text, color depth or compression ratio of image, etc.

For a Web page P consisting of several blocks, the resulting information fidelity is defined as a weighted sum of the information fidelity of all blocks in P. Straightforwardly, we can employ the importance values from DRESS as the weights of contributions to the perceptual quality of the whole page. Thus, the information fidelity of an adapted result is described as

$$IF(P) = \sum_{B_i \in P} IMP_i \cdot IF_{B_i} \tag{2}$$

This is used as the object function of our adaptation algorithm. In this paper, we assume *IF* value only depends on the version of content block, i.e., whether it is summarized or not. If a content block is replaced by its alternative, the *IF* value of this block is defined to be 0, otherwise it is 1.

4.2 Problem Definition

We introduce P' as the set of unsummarized information blocks in a Web page P, $P' \subset P = \{B_1, B_2, ...B_N\}$. Thus, our misson is to find the block set P' that carries the largest information fidelity while meets the display constraints.

In order to ensure that all the content blocks are possible to be included in the final presentation, the following constraint should be satisfied.

$$\sum_{B_i \notin P'} size(ALT_i) + \sum_{B_i \in P'} MPS_i \le Area \tag{3}$$

where *Area* is the size of target area and *size (x)* is a function which returns the size of display area needed by ALT_i. If the constraint (3) is transformed to

$$\sum_{B_i \in P'} (MPS_i - size(ALT_i)) \le Area - \sum_{B_i \in P} size(ALT_i) \tag{4}$$

then the Web layout optimization problem becomes:

$$\max\left(\sum_{B_i \in P} IMP_i \cdot IF_{B_i}\right) = \max_{P'}\left(\sum_{B_i \in P'} IMP_i\right) \quad \text{subject to} \quad \sum_{B_i \in P'} (MPS_i - size(ALT_i)) \le Area - \sum_{B_i \in P} size(ALT_i) \tag{5}$$

We can see that problem (5) is equivalent to a traditional NP-complete problem, 0-1 knapsack. The constraint (4) also implies that the display size should not be too small, otherwise we will not be able to find a valid layout even when all the blocks have been summarized. In this rare case, sub-tree summarization will become necessary.

Since constraint (3) does not ensure that the *MPH* or *MPW* will be satisfied, we solve the problem by a two-level approach. First we use a branch and bound algorithm to enumerate all possible block set P', then for each block set, we use a capacity ratio based slicing algorithm to test whether a valid layout can be found. By this process, we search among all possible block set P' to select the optimal one, i.e. the scheme that achieves the largest *IF* value while presents an aesthetic view.

4.3 Page Rendering Algorithm

Since problem (5) is NP-complete, it is very time-consuming to simply try possible solutions one by one. We design a branch-and-bound algorithm to select the block sets efficiently.

4.3.1 Block Set Selection

As shown in Figure 2, we build a binary tree for block set selection in which the root node is a null set Φ implying all the blocks are summarized and

- Each node denotes a set of unsummarized blocks;
- Each level presents the inclusion of an information block;
- Each bifurcation means the choice of keeping original contents or making summary of the block in the next level.

Thus, the height of this tree is *N*, the total number of blocks inside the Web page, and each leaf node in this tree corresponds a different possible block set P'.

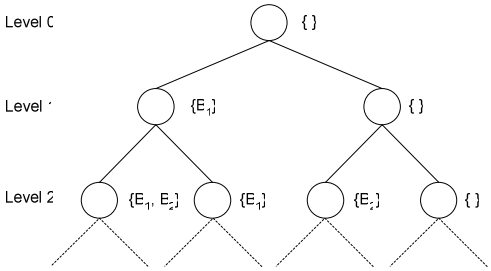


Figure 2. The binary tree used for selecting optimal block sets.

For each node in the binary block tree, there is a boundary on the possible *IF* value it can achieve among all of its sub-trees. Obviously, the lower boundary is just the *IF* value currently achieved when none of the unchecked blocks can be added, that is, the sum of *IF* values of blocks included in the current configuration. And the upper boundary is the addition of all *IF* values of those unchecked blocks after the current level, in other words, the sum of *IF* values of all blocks in the Web page except those summarized before the current level. We perform a depth-first traversal on this tree according to following constraints:

- Whenever the upper bound of a node is smaller than the best *IF* value currently achieved, the whole sub-tree of that node including itself will be truncated.
- At the same time, for each node we check area size constraint in Equation (3) to verify its validity. If the constraint is broken, the node and its whole sub-tree will be

truncated, because including a new block will increase the sum of *MPS* values.

l If we arrive at a block set with an *IF* value larger than the current best *IF* value, we will check the feasibility of this solution by running the layout algorithm in Section 4.3.2. If this solution is feasible, we will replace the current best *IF* value by this one.

By checking both the bound on possible *IF* value and the layout validity of each block set, the computation cost is greatly reduced. We also use some other techniques to reduce the time of traversal such as arrange all the blocks in a decreasing order of their importance values at the beginning of search, since in many cases only a few blocks contribute the majority of *IF* value.

4.3.2 Capacity Ratio Based Slicing

When checking the feasibility of a block set *P'*, we try to find an aesthetic layout to put all content blocks and summarizations in the target area. The general problem of Web layout optimization is similar to the floor planning in VLSI design, which is known as a NP-hard problem. However, as mentioned before, because the slicing tree structure remains constant during our adaptation process, so only the slicing numbers need to be decided. We employ a capacity ratio based slicing method to deal with this problem in two steps as shown in Figure 3. First, we go through the slicing tree in a bottom-up way to calculate the capacity, height and width constraints for each node. Then, the slicing numbers are computed in a top-down way.

```
Slicing ()
{
   // calculate capacity, height and width constraints
   Aggregate (root);

   // calculate slicing numbers
   root->area = TARGET_AREA;  // target display size
   root->height = TARGET_HEIGHT; // target display height
   root->width = TARGET_WIDTH;  // target display width
   if (Allocate (root) == true) return true;
   else return false;
}
```

Figure 3. Capacity ratio based slicing algorithm.

The detailed flow of first step is shown in Figure 4, where the capacity, minimal height and minimal width of each node stands for its display constraints. For a summarized block, they correspond to the values of its alternative which can be calculated automatically from the summary text.

The second step compares the capacities of two sub-trees of each inner node and let the slicing number of that node be proportional to them. This leads to the name of our algorithm, capacity ratio based slicing. We also perform some adjustments on the slicing numbers to meet the display constraints. Figure 5 presents the algorithm details.

The complexity of this layout algorithm is *O(N)*, since both steps can be performed in *O(N)* time. Therefore, our capacity ratio based slicing algorithm is fast enough to check the layout feasibility of each possible block selection. An example is shown in Figure 6. The *MPS* of the four content blocks in the page are 1000, 1000, 2500 and 2500, respectively. Suppose all of them are not summarized in the adaptation, the slicing number of the root will be: 1000/(1000+1000+2500+2500) =0.14. That is to say, if the screen is 600 in height, we should split the region horizontally at the height of 600*0.14=84. This procedure is executed recursively until all the slicing numbers are decided. However, if block *1* has set its *MPH* to 100, then we will split the display at the height of 100 instead in order to meet this requirement.

```
Aggregate  (p )
{
   if p is a leaf node
      if p is a summarized block
         p–>capacity = size (p->alternative);
         p–>minheight = height (p->alternative);
         p–>minwidth = width (p->alternative);
      else  // p is unsummarized
         p–>capacity = p->MPS;
         p–>minheight = p->MPH;
         p–>minwidth = p->MPW;
   else // p is an inner node
      Aggregate  (p->lchild );
      Aggregate ( p->rchild );
      p–>capacity = p->lchild–>capacity + p->rchild–>capacity;
      if p->label == vertical
         p–> minheight = max(p->lchild–>minheiht, p->rchild–>minheight );
         p–> minwidth = p->lchild–>minwidth + p->rchild–>minwidth;
      else // p's label is horizontal
         p–> minheight = p->lchild–>minheight + p->rchild–>minheight;
         p–> minwidth = max (p->lchild–>minwidth, p->rchild–>minwidth);
}
```

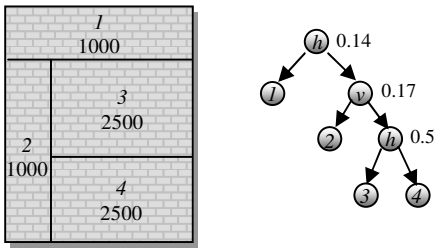
Figure 4. The algorithm for calculating the capacity, height and width constraints of each node.

```
Allocate (p)
{
   if p–>area < p–>capacity or p–>height < p–>minheight
                                          or p->width < p->minwidth
      return false;

   if p is an inner node
      p->slicingnumber = p->lchild–>capacity / p–>capacity;
      if p->label == vertical
         p->lchild–>height = p->height
         p->rchild–>height = p–>height;
         p->lchild–>width = p–>slicingnumber * p–>width;
         p->rchild–>width = p–>width – p->lchild–>width;
         if p->lchild–>width < p->lchild–>minwidth
            p->lchild–>width = p->lchild–>minwidth;
            p->rchild–>width = p->width – p->lchild–>minwidth;
         else if p->rchild–>width < p->rchild->minwidth
            p->rchild–>width = p->rchild–>minwidth;
            p->lchild–>width = p–>width – p->rchild–>minwidth;
      else // p's label is horizontal
         p->lchild–>width = p->width
         p->rchild–>width = p–>width;
         p->lchild–>height = p–>slicingnumber * p–>height;
         p->rchild–>height = p–>height– p->lchild–>height;
         if p->lchild–>height < p->lchild–>minheight
            p->lchild–>height = p->lchild–>minheight;
            p->rchild–>height = p->width – p->lchild–>minheight;
         else if p->rchild–>height < p->rchild->minheight
            p->rchild–>height = p->rchild–>minheight;
            p->lchild–>height = p–>height– p->rchild–>minheight;

      if (Allocate (p->lchild ) == false) return false;
      else if (Allocate (p->rchild ) == false) return false;
      else return true;
   else return true // do nothing when p is a leaf node
}
```

Figure 5. The algorithm for assigning the final display rectangle to each node.



(a) (b)

Figure 6. (a) An example Web layout with *MPS* labeled. (b) The corresponding slicing tree with slicing number labeled.

4.3.3 Content Accommodation

After layout optimization, each block, whether summarized or not, has been assigned a rectangle region for display. According to the slicing algorithm, the area size of the assigned region for each node will be larger than its *MPS* value. The height and width of the display region will also meet the requirements of each block. We use the *ADJ* attribute of each information block to aid the content accommodation process. If a block is indicated to be adjustable, we will fit it into the target rectangle by zooming and wrapping. Otherwise, we will only zoom the block while maintaining its aspect ratio. Other content adaptation techniques like attention model based image adaptation [7] can also be integrated into this step. Sometimes, the size of summarized alternatives may be much smaller than their allocated region. Therefore, we extract some keywords from original contents and append them in the unoccupied space to throw more light.

When a user follows the link of an alternative, a technique similar to the fisheye view [17] is adopted to highlight the corresponding block. We redo the adaptation process with a new constraint that this block should not be summarized. However, if no such solution is found, the user will be navigated to a new page of the original contents which is displayed using the whole target area. Note if the block is oversize, scrolling is inevitable. Therefore, we suggest huge blocks be avoided in the Web design.

5. DRESS-BASED WEB BROWSING ON SMALL TERMINALS

We have implemented a prototype of DRESS-based Web browser to validate the performance of our proposed scheme. With DRESS and the page rendering algorithms, the browser can dynamically select and summarize proper parts of Web pages, then optimize the layout for target area, and finally adapt block contents to fit into the corresponding regions allocated. Experimental results are very encouraging.

5.1 Implementation of DRESS Browser

In our prototype, the DRESS structure is stored as comments within the HTML files or style sheets. For example, the slicing tree structure of the Web page in Figure 1 is saved as a Polish expression "((A – (B | (C | (D – E)))) – F)" where "–" and "|" denote horizontal and vertical slicing, respectively. Other attributes of DRESS are described in the same way as shown in Figure 7. On second thoughts, by adopting similar annotation mechanism described in [14], we can also store the DRESS information into an external XML document which uses XPath and XPointer to indicate the information blocks in the original HTML file.

We allow user to designate the desired target area in the prototype which was developed based on Microsoft Internet Explorer. When browsing a Web page, our prototype first parses the DRESS information inside, and then generates a main result HTML file that exactly fits the target display. At the same time, those summarized block contents are saved in temporary HTML files which are accessible by following the links of alternatives. These intermediary data can be saved in memory instead of files if the adaptation is performed at client side.

```
<!-- DRESS SlicingTree = ( ( A – ( B | ( C | ( D – E ) ) ) ) – F ) -->
<!--Block ID="A" IMP="0.15" MPS="35000" MPH="80" MPW="400"
          ADJ="No" ALT= "Google News Search" -->
 contents of block A …
 …
<!--Block ID="B" IMP="0.1" MPS="14000" MPH="140" MPW="100"
          ADJ="No" ALT= "Left Sidebar" -->
 contents of block B …
 …
<!--Block ID="C" IMP="0.4" MPS="80000" MPH="150" MPW="200"
          ADJ="Yes" ALT= "Top Stories" -->
 contents of block C …
 …
<!--Block ID="D" IMP="0.2" MPS="30000" MPH="100" MPW="150"
          ADJ="Yes" ALT= "Hot News List" -->
 contents of block D …
 …
<!--Block ID="E" IMP="0.1" MPS="20000" MPH="100" MPW="200"
          ADJ="No" ALT= "In the news" -->
 contents of block E …
 …
<!--Block ID="F" IMP="0.05" MPS="9000" MPH="40" MPW="150"
          ADJ="Yes" ALT= "Google Footer" -->
 contents of block F …
 …
<!-- DRESS End -->
```

Figure 7. An example DRESS representation.

It is worth noticing that DRESS does not require additional modification on client devices. The transformation from a DRESS document to normal HTML document can be done at either the content server or intermediary proxies. The process will be just like the deployment strategy of XML plus XSLT currently. Thus, it can be deployed incrementally on the Internet.

5.2 Experimental Results

We carried out an experiment to compare our DRESS-based approach with the widely used thumbnail method. To illustrate our scheme, we use the same example page in Figure 1 whose DRESS is defined in Figure 7.

Figure 8 shows the comparison results on three typical screen sizes without scrolling. As shown in Figure 8(a, c, e), when using Web thumbnails on Handheld PC (640x240), Smartphone (128x160), and Pocket PC (240x320), almost all the contents are hardly recognizable because of excessive down-sampling. In contrast, our solutions in Figure 8(b, d, f) provide a much better experience in both content reading and link navigation. For instance, in Figure 8(b) and Figure 8(f), parts of the unimportant or oversize page contents are summarized while the rest parts are still kept readable in original version to deliver the maximal information fidelity. If a user clicks the link of left sidebar in Figure 8(f), a new layout is generated with the sidebar expanded as shown in Figure 8(g). In Figure 8(d), the target area is too small to accommodate any unsummarized information block, thus all blocks are presented in summaries as a table-of-content. Note in Figure 8(h) the rotation factor, which is common among mobile devices, is taken into consideration and a better result is presented accordingly with the left navigation sidebar added.

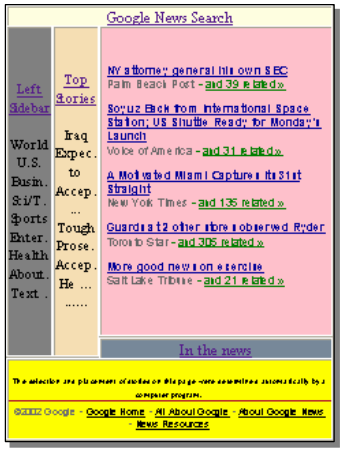
(a) Thumbnail view on Handheld PC

(b) Adapted result on Handheld PC

(c) Thumbnail view on Smartphone

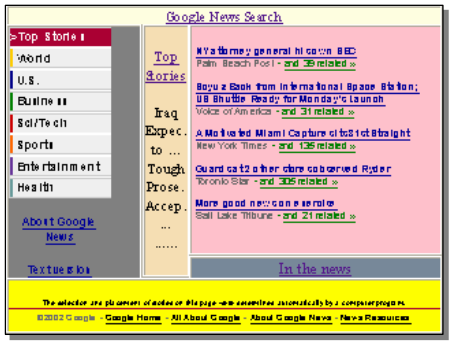
(d) Adapted result on Smartphone

(e) Thumbnail view on Pocket PC



(f) Adapted result on Pocket PC

(g) A new layout after clicking the left sidebar in (f)



90°

(h) Adapted result on Pocket PC using a rotated display

Figure8. The results of DRESS-based layout adaptation compared with thumbnail views.

Another experiment was also conducted to test the efficiency of our algorithm by logging the computational time costs while making 10 times layout adaptation for each Web page. We included the time cost for the procedures of both block selection and layout determination. Our test bed is a Dell Optiplex GX 240 using a 1.8GHz Pentium 4 processor, 512 MB of RAM, and Windows XP Professional system. Our test data of 16 Web pages were collected from the most popular Websites such as MSN, Yahoo!, Google. We manually added the DRESS structure into these pages, among which the information block numbers vary from 5 to 20. In the test we got an average time cost at 18 microseconds, i.e. about 55,000 pages per second, with variation from 2 to 56 microseconds. The result shows that without code optimization our algorithm is already fast enough to be employed in real-time Web browsing systems on either Website servers or proxy servers, or even on client devices.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a *Document REpresentation for Scalable Structure (DRESS)* to help information providers to make composite documents, typically Web pages, scalable to display size in both logic and layout structure. By integrating both techniques from computer aided design and information extraction, DRESS allows easy negotiation between author and viewer, hides layout manipulation from author but still keeps the result structure predicable, and represents contents in dynamic layouts to fit various user preferred display sizes.

Currently, we are developing a Web authoring tool to create and insert DRESS structure with ease of use. Though we focus on a new document representation, automatic transformation from existing Web contents is also crucial to its incremental deployment. We are looking forward to automatic DRESS generation by leveraging our previous experience on Web page analysis. DRESS-based Web personalization is also one of our future directions.

7. ACKNOWLEDGMENT

We would like to express our special appreciation to Xin Fan, Xiaodong Gu and Yu Chen for their insightful suggestions.

8. REFERENCES

[1] Badros, G., Borning, A., Marriott, K., and Stuckey, P. Constraint Cascading Style Sheets for the Web. Proc. UIST'99 (Asheville, USA, Nov 1999), 73-82.

[2] BickMore, T.W., and Schilit, B.N. Digestor: Device-Independent Access to the World Wide Web. Proc. WWW'97 (Santa Clara, USA, April 1997), 655–663.

[3] Björk, S., Holmquist, L.E., Redström, J., Bretan, I., Danielsson, R., Karlgren, J., and Franzén, K. WEST: A Web Browser for Small Terminals. Proc. UIST'99 (Asheville, USA, Nov 1999), 187-196.

[4] Borning, A., Lin, R.K., and Marriott, K. Constraint-Based Document Layout for the Web. ACM Multimedia Systems Journal, Vol. 8, No. 3, 2000, 177-189.

[5] Buyukkokten, O., Garcia-Molina, H., Paepcke, A., and Winograd, T. Power Browser: Efficient Web Browsing for PDAs. Proc. CHI'00 (Hague, Netherlands, April 2000), 430-437.

[6] Chen, J.L., Zhou, B.Y., Shi, J., Zhang, H.J., and Wu, Q.F. Function-based Object Model towards Website Adaptation. Proc. WWW'01 (Hong Kong, China, May 2001), 587-596.

[7] Chen, L.Q., Xie, X., Ma. W.Y., Zhang, H.J., and Zhou, H.Q. Image Adaptation Based on Attention Model for Small-form-factor Devices, To appear in the 9th International Conference of Multimedia Modeling, Jan 2003.

[8] Cohoon, J.P., and Paris, W.D. Genetic placement. IEEE Trans. on Computer-Aided Design, No. 6, 1987, 956-964.

[9] Fuchs, M. An Evolutionary Approach to Support Web Page Design. Proc. 2000 Congress on Evolutionary Computation (Piscataway, USA, 2000), 1312-1319.

[10] González, J., Rojas, I., Pomares, H., Salmerón, M., Prieto, A., and Merelo, J.J. Optimization of Web Newspaper Layout in Real Time. Computer Networks, Vol. 36, Issue 2-3, July 2001, 311-321.

[11] González, J., Rojas, I., Pomares, H., Salmerón, M., and Merelo, J.J. Web Newspaper Layout Optimization Using Simulated Annealing. IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics, Vol. 32, No. 5, Oct 2002, 686-691.

[12] Gu, X.D., Chen, J.L., Ma, W.Y., Chen, G.L. Visual Based Content Understanding towards Web Adaptation. 2nd Intl. Conf. on Adaptive Hypermedia and Adaptive Web Based Systems (Malaga, Spain, May 2002), 164-173.

[13] Han, R., Perret, V., and Naghshineh, M. WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing. Proc. CSCW'00 (Philadelphia, USA, Dec 2000), 221-230.

[14] Hori, M., Kondoh, G., Ono, K., Hirose, S., and Singhal, S. Annotation-based Web Content Transcoding. Computer Networks, Vol. 33, No. 1-6, 2000, 197-211.

[15] Milic-Frayling, N., and Sommerer, R. SmartView: Flexible Viewing of Web Page Contents. Proc. WWW'02 (Honolulu, USA, May 2002).

[16] Opera Small Screen Rendering. http://www.opera.com/products/smartphone/smallscreen/

[17] Sarkar, M. and Brown, M.H. Graphical Fisheye Views. Communications of the ACM, Vol. 37, No. 12, 1994, 73-84.

[18] Wobbrock, J.O., Forlizzi, J., Hudson, S.E., and Myers, B.A. WebThumb: Interaction Techniques for Small-Screen Browsers. Proc. UIST'02 (Paris, France, Oct 2002), 205-208.

[19] Xie, X., Zeng, H.J., and Ma, W.Y. Enabling Personalization Services on the Edge. To appear in ACM MM 2002 (Juan-les-pins, France, Dec 2002).

[20] Yang, Y.D., and Zhang, H.J. HTML Page Analysis Based on Visual Cues. Proc. ICDAR'01 (Seattle, USA, Sep 2001).