IEEE distributed systems ONLINE
Expert-authored articles and resources

October 2003

From the October issue
Story location: http://dsonline.computer.org/0310/f/leep.htm

Web Browsing on Small Displays

Dik Lun Lee, Ka Kit Hoi, and Wing Sing Dicky Wong • **Hong Kong University of Science and Technology**
Jianliang Xu • **Hong Kong Baptist University**

Mobile computing has attracted attention in the past few years owing to technology advances. However, viewing documents on a mobile device isn't easy because of the limited display size, CPU power, and bandwidth. This is an obstacle for wireless Web browsing because there is no automatic transformation of HTML[1] documents designed for display on large screens to small PDA screens.

The same problem occurs in devices for Web access (or Web-like access) on television. Although televisions are generally larger than most computer monitors, the relatively low resolution and long viewing distance reduces the amount of information they can display. In this article, *small displays* are devices that display a small amount of information for comfortable user viewing, because either the screens are physically small or they have very low resolution.

Small displays increase the burden on Web page authors and application designers because they must consider all possible display capabilities of user devices when creating a Web page. To ease such tasks, this article presents an automatic document segmentation and presentation system. The DSPS has three primary functions. First, it automatically divides a Web document into different logical segments on the basis of the device's display size and the document's hierarchical structure and content. Second, it extracts the summary and overview information from the logical segments to help users locate relevant information. Third, it creates an interface for clear, user-friendly presentation of the segments for rapid access to the desired information.

Information rendering on small displays

The obvious solution to present information on small-display devices is tailoring it for each type of device. Several information providers manually create documents for specific mobile devices so that users can download them into their devices (see the "Useful URLs" sidebar). The major problem with tailor-made information is portability. Often, several versions of a document are needed for different types of user devices.

Certain Web browsers can run on handheld devices and WebTV devices[2] (see the "Useful URLs" sidebar). These Web browsers usually work by filling up the screen line by line. This provides a good solution for simple Web pages that don't use tables and frames as formatting tools. However, when a Web page contains tables and frames—which are common on today's Web—the Web page layout in those browsers becomes messy and hard to read. The main problem is that the browsers don't perform intelligent content transformation to make the information suitable for display on small screens.

Small displays and user interaction

Some studies have shown that a small display doesn't dramatically impact readers' comprehension level or reading speed. [3,4,5] However, heavy interactions such as scrolling annoyed many users. Users had to take more time and keystrokes to get their desired information.

Matt Jones and his colleagues conducted an experiment on small displays' impact.[6] Twenty computer science staff and student volunteers viewed a Web site with a design common to many commercial sites. The experiment revealed interesting user behavior: large-screen users showed a greater tendency to follow navigation paths, while small-screen users had much shorter navigation paths.

Such a finding illustrates some principles of designing or creating Web pages for small-display devices. First, efficient access to the information items and search functions play an important role. Second, the information layout should reduce navigation operations such as scrolling. Finally, a Web site's summary information should be presented before the detailed contents.

Structured data tree and content tree

To process HTML documents, the DSPS must have a document model. Seung-Jin Lim and Yiu-Kai Ng proposed the *structured data tree* and *content tree* to represent the hierarchical information of HTML documents.[7,8]

An SDT (see Figure 1a) is a type of parse tree for an HTML document based on the HTML grammar. A CT captures the hierarchical relationships among HTML document contents. It's a tool for representing and analyzing a document's internal structure. Figure 1b shows a CT constructed from the SDT in Figure 1a.

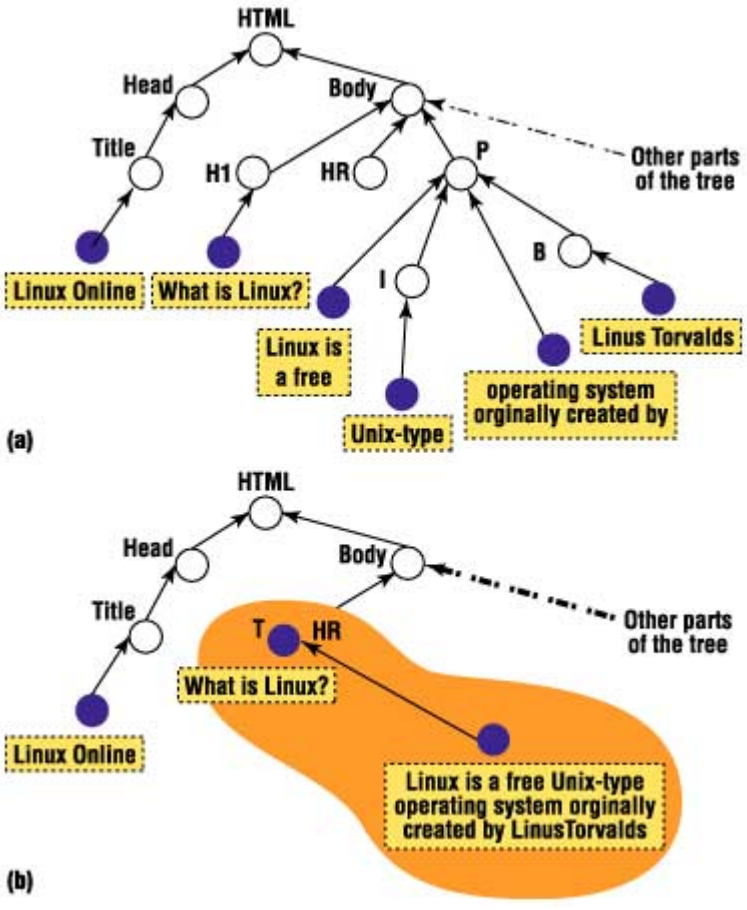

Figure 1. Two tree structures: (a) structured data tree; (b) content tree

However, Lim and Ng's algorithm for constructing CTs considered only limited HTML tags such as *H*1, *TITLE*, and *BODY*. Tables and frames, which are heavily used in formatting Web pages, are not included. We extend their algorithm to tables and frames.

Converting an HTML document to content tree

An HTML document is semistructured because the HTML grammar implies document's high-level structure. To obtain such structural information of an HTML document, we use a series of algorithms and heuristic methods to extract the document structure (expressed in a CT).

Figure 2 provides an overview of converting an HTML document *D* to a structured document D_s that can be represented by a CT, CT_{DS}. The CT_{DS} should be able to facilitate the presentation of *D* on a small display screen *Scr*(*w*, *l*), which can display *l* lines, each line consisting of up to *w* characters.

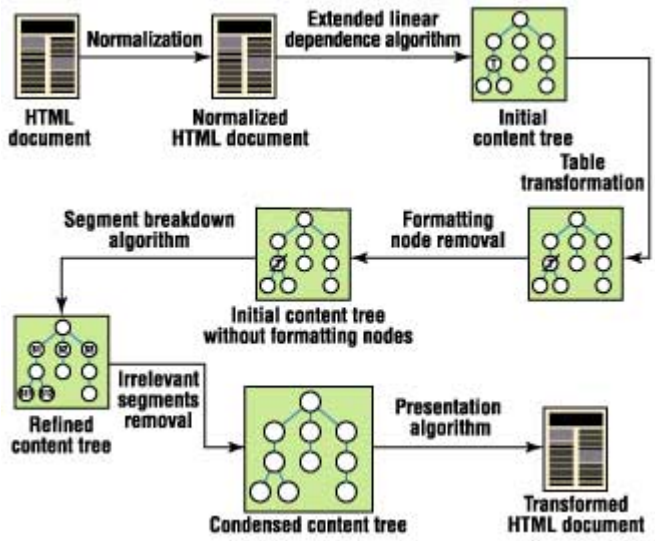

Figure 2 The conversion from HTML document to content tree.

An HTML parser converts and normalizes an HTML document into an intermediate CT. The intermediate CT is then converted into an initial CT using the *extended linear dependence algorithm*. Then, the *table transformation algorithm* transforms table structures in the HTML document into different subtrees. The *formatting node removal algorithm* further refines the CT structure by analyzing each node's formatting style. To customize the display, the *segment breakdown algorithm* breaks down the CT's flat structure into a hierarchical structure by incorporating two factors: the formatting values of all the text segments and the display's size. Some irrelevant subtree segments are then removed from the CT by the *irrelevant segments removal algorithm*. Finally, the resulting CT is transformed into a form that suitable for presentation on the screen.

Classification of HTML tags

The HTML tags are classified into three types:

- *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. On the basis of their semantic meanings, w can assign linear dependence order to them: H3 is dependent on H2, and H2 is dependent on H1.
- *Block tags* should be treated as individual entities. For example, TABLE, TR

and TD are 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 be further classified into *block begin tags* and *block element tags*. Block begin tags (for example, TABLE) indicate the beginning of a block entity construct. Block element tags (for example, TR and TD) are the building elements of the block entity.

- *Formatting tags* are used in formatting the text rather than indicating the structural information. These tags are an attribute of a piece of text rather than a node in the CT. These tags provide useful information in the transformation process.

Table 1 shows the classification of tags. HTML tags that aren't included in the table are discarded to simplify the implementation.

Table 1. Classification of HTML tags.

Tag type	HTML tags
Structural	Title, H1, H2, H3, H4, H5, H6, P, BR
Block begin Block element	TABLE, UL, OL TH, TR, TD, LI
Formatting	TT, I, B, U, BIG, SMALL, EM, STRONG, FONT, A

Normalization of the HTML document

To simplify the structural analysis, an HTML document is preprocessed and normalized into a simple, clean form. According to HTML grammar, formatting tags have few restrictions on where they can appear. They can enclose a whole table or exist in the table cells. Thus, each HTML document is normalized so 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.

Extended linear dependence algorithm

The extended-linear-dependence algorithm is based on the CT construction algorithm proposed by Lim and Ng.[7,8] The CT construction is based on the dependence of tags. The linear dependence defines a sequence of tags (see Table 2) that indicates the tags' conceptual, structural, and scope meaning. Any access path in the CT from the root node to any leaf node should follow the linear dependence.

Table 2. The dependence order of HTML tags.

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

We extend Lim and Ng's algorithm to construct an initial CT from a normalized HTML document (that is, an intermediate CT, CT_{SDT}). We employ different construction strategies for different types of tags (that is, *structural*, *block*, and *formatting tags*). We 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 traversal includes the block begin tags. The extended-linear-dependence algorithm, which takes the intermediate CT_{SDT} as the input, works as follows:

1. Create a list of nodes L_{CTSDT} with block elements restricted in-order traversal on the root n_r.
2. Create a content subtree root node sroot = (*nr.tag, “”*).
3. Create an empty stack *BS*.
4. Set *currnode* to *sroot*.
5. For each node cn_i in L_{CTSDT}
    1. Create a node n_i (cn_i tag “”)
    2. If cn_i is a leaf node, set n_i.text = cn_i.body.
    3. 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*
    4. If $cn_i.tag$ is a structural tag or formatting tag and its dependence value is larger than that of *currnode.tag*— that is, $cn_i.tag$ depends on *currnode.tag*—attach n_i to *currnode* as a child, and set *currnode* = n_i.
    5. Otherwise, set *currnode* = the parent node of *currnode*, and repeat Step d.
    6. For each pair (*cn*, *n*) in *BS*, apply the extended linear dependence algorithm on *cn* and attach the returned subtree to *n*
    7. Return the subtree rooted at *sroot*.

Figure 3 shows an example of an initial CT.

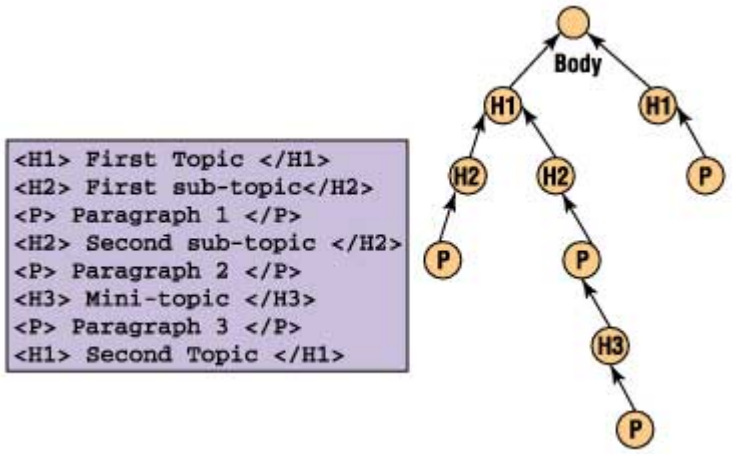

Figure 3. An example of extended-linear-dependence algorithm.

Table transformation

Tables are commonly used in formatting Web pages because other HTML tags provide insufficient formatting power. However, this complicates extracting a Web page’s logical structure. We assume that tables contain some textual content. Tables containing figures can only be identified at the parsing stage using heuristic measures. So, each table can be transformed into a single node when we construct the initial CT. We also assume that the table has no column or row span. If it does use column or row spans, the table is transformed into a normal table by splitting the spanning columns or rows and duplicating the spanning cells’ contents accordingly.

Another assumption for the table transformation is that the text segments belong to the same level of the logical hierarchy and don’t span across different columns.

For example, if all the text segments belong to a section, all of them will be located in one column. As such, the logical hierarchy won't break when the table transformation separates two adjacent columns. This assumption is valid for most Web pages because HTML documents don't have length limits. Therefore, splitting a logical paragraph into two columns isn't necessary.

On the basis of the above assumptions, we transform a table as follows. Let $T(m,n)$ denote a table of m columns and n rows; each cell of the table is represented by $c(i, j)$, where $1<i<m$ and $1<j<n$. Let $S_{(i,j)}$ denote 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 table transformation algorithm:

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

The algorithm classifies each column of a table as a separate logical unit. It also assumes that cells in the same column are highly related to each other but less related to the cells in other columns.

Removing formatting nodes

A node is defined as a formatting node if it's a formatting tag. Formatting nodes complicate the structural analysis because a heavily formatted text segment would produce a large subtree in the CT. To facilitate the heuristic-breakdown algorithm, these types of nodes are removed. We introduce the concept of *formatting value*, which can simplify a complicated subtree structure into a single text node together with a quantitative measure of the formatting style.

Different formatting tags emphasize effects to different degrees. 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 greater emphasizing effect. Table 3 shows the formatting value of each formatting tag in our implementation. The value is derived by estimating each formatting tag's emphasizing effect.

Table 3. The formatting value of formatting tags.

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	--	--

The formatting-nodes-removal algorithm uses a numeric value to replace a collection of formatting nodes. The numeric value reflects how heavily the text is formatted. It's calculated by adding the formatting values of the formatting tags applied to the text. Figure 4 shows an example of the formatting nodes removal algorithm.

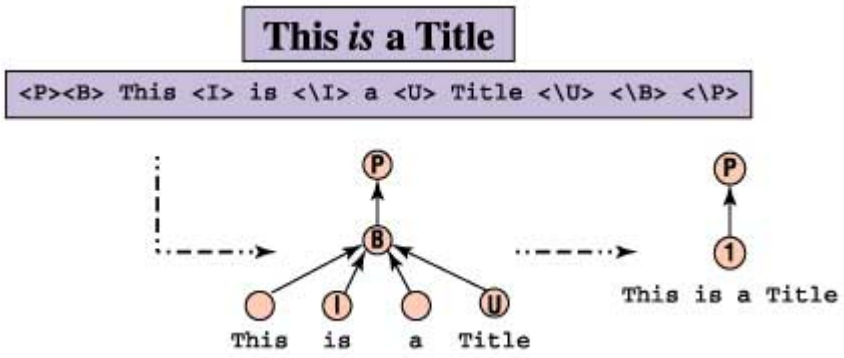

Figure 4. A formatting-nodes-removal algorithm.

Segment breakdown algorithm

The segment breakdown algorithm takes the display parameters as the input and tailors the information for appropriate display. The algorithm divides an HTML segment by identifying the titles in the segment and using them as the cutting points to create subsegments. The segment breakdown algorithm can be applied to each HTML segment until a subegment's size is small enough for display.

The segment breakdown algorithm identifies the highly formatted text nodes and promotes them to a higher hierarchical level of the CT. Less-formatted texts 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 for visualizing the document on small-display devices because the document fragments represented by the flat tree structure can be replaced by several logically separated segments. An overview of these segments, which consists of most highly formatted texts, is also generated.

Removing irrelevant segments

Some irrelevant segments in HTML documents (for example, ads) are common, especially for commercial Web pages. Irrelevant segments can be eliminated so that the HTML document can be visualized on small display devices effectively. To do so, two assumptions are made for irrelevant segments. First, irrelevant segments are usually small and second, irrelevant segments have few or no keywords in common with the relevant segments in the same logical level.

We use the Boolean vector space model to calculate two texts' similarity scores. Let $(w_1, w_2, \ldots w_n)$, be the keyword space and $v_t(b_1, b_2, \ldots b_n)$ be the bit vector of text t such that

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

Given two text vectors v_1 and v_2, we define the *similarity score* as

$$SIM(v_1, v_2) = \frac{|v_1 \cdot v_2|}{\sqrt{|v_1| + |v_2|}}.$$

Using the vector's cardinality as the divisor minimizes the effect of a long

paragraph, especially in intradocument analysis. This is because most entities in a document are titles, subtitles, and short paragraphs, which will be unfairly treated if we do not normalize them based on the basis of segment size.

For a node *n* in a CT, we define Seg_n as the text of the segment represented by the subtree rooted at *n*,and define n_i as a node in the subtree. The algorithm for calculating the relevance score *RSni* of a node n_i is:

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

Subtrees with relevance scores lower than a threshold *tr* are considered irrelevant and can be purged from the original CT. 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. So, we can use a single threshold value for different levels of the logical hierarchy. The algorithm's basic idea is that a segment is irrelevant if it isn't similar to the rest of the text that's in the same level of the logical hierarchy.

Something interesting happens when we apply the irrelevant-segment-removal algorithm to subtrees with few siblings. Siblings' scores would be low if the number is too small or they have no common keywords. For example, if two children of a node have no common keywords, both children's scores are zero. The algorithm cannot conclude which child is irrelevant. So, the algorithm won't remove any children if all the children have scores lower than a minimum threshold.

The DSPS

We developed the DSPS using the Java 2 Platform. The Java 2 Platform contains a robust HTML parser and supports both XML and the Document Object Model. Because most of the operations in the DSPS interact with tree structures such as SDTs and CTs, XML and DOM support is essential. A clear mapping exists from a tree structure to an XML document or DOM structure.

Internally, the DSPS consists of the HTML document transformation subsystem and the document presentation subsystem. The DSPS takes two inputs: the target HTML document's URL and the display's configuration information. A transformed HTML document is generated after a series of operations, as we discussed in the last section.

Case study: IEEE DS Online

Figure 5 demonstrates how the DSPS converts normal HTML pages into pages suitable for small displays. Owing to space limitations, we show one example using the *IEEE DS Online* site (http://dsonline.computer.org). More examples appear elsewhere.[9]

Figure 5. A CT of the front page of *IEEE DSO*.

IEEE DS Online's main page has three main segments (indicated by the arrows in Figure 5): the top banner (represented by the subtree rooted at "Published by the IEEE..." in the CT), the middle part (represented by the subtree rooted at "site sponsors"), and the bottom part (represented by the subtree rooted at "DS Online ISSN: 1541 ..."). The top banner and the bottom part contain information such as archives, ISSN number, and contact. Their similarity scores are 0.87 and 0.81, respectively. On the contrary, the middle part, as the page's main body has a higher similarity score, 1.0.

The main body uses a table to format its content. The left side has several ads for ordering IEEE magazines. The center area contains a collection of feature items, while the right side lists the topic areas. These three sections are three columns in a table. This coincides with our assumption that text segments belong to the same level of the logical hierarchy and don't span across different columns. The similarity scores for these three components are 0.73, 1.0, and 0.94, respectively, indicating that the left side is less relevant to the main content. So, if we set the relevance threshold to 0.9, the mobile device will show only the center part and the main body's right side, which is satisfactory.

In the CT, “News” is at the same level as “*IEEE Pervasive Computing*,” whereas “Works in Progress” is one level lower. This is because HTML documents lack semantic information and our document structure analysis is based on segment formats. In the example, there is a <P> </P> pair embracing “*IEEE Pervasive Computing*” and “Works in Progress,” and another embracing “News.” This misleads our system that “Works in Progress” is under “*IEEE Pervasive Computing*.”

With the CT refined by removing irrelevant segments, we can display the Web page on a small display on a segment-by-segment basis. Figure 6a shows that overview of this page on a Palm m505 after the user requests this URL, where the relevance threshold is set at 0.5. When the user taps on the title “Gregory D. Abowd,” the corresponding block appears on the screen (See Figure 6b).

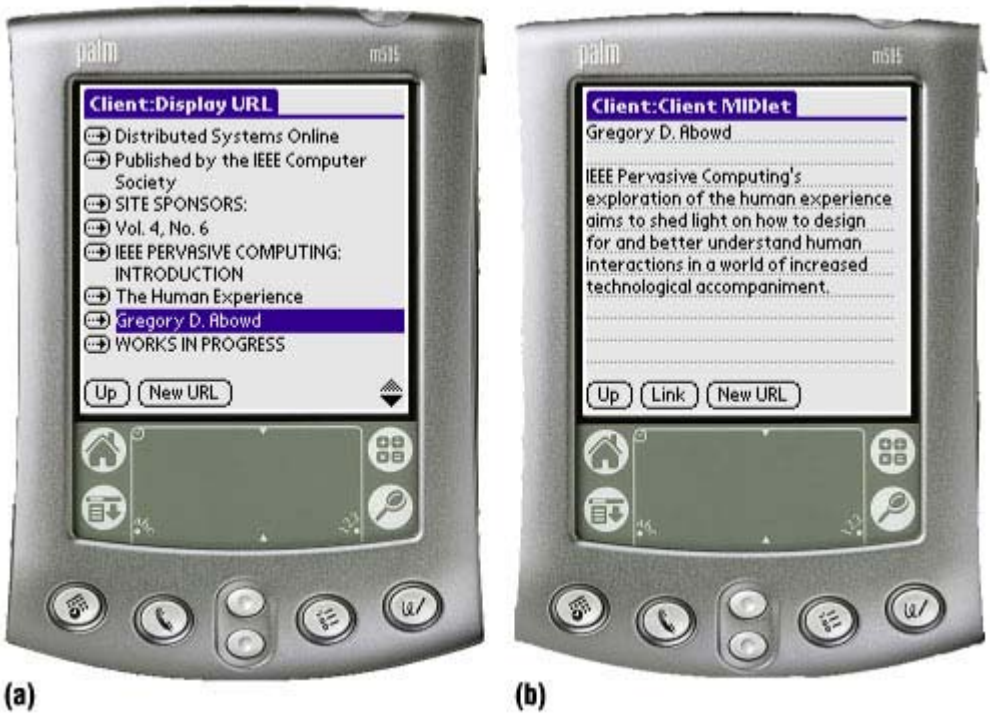

Figure 6: An overview of *IEEE DS Online* when the client requests the related URL; (b) the result displayed when the user selects “Gregory D. Abowd.”

The DSPS is targeted for real-world, commercial HTML documents that contain many screen-oriented formatting styles such as tables and complicated text-formatting structures. We’re extending the system to perform more intelligent content analysis based on user profile specified in the CC/PP protocol.

Useful URLs

- Microsoft WebTV (www.webtv.com) provides Web-based access services through televisions.
- PocketInfo (www.pocketinfo.com) is a public domain initiative that provides information for users of handheld computers, palmtops, and organizers.
- AvantGo (www.avantgo.com) delivers information to PalmOS devices, PocketPCs, and next generation phones.
- Netfront (www.access.co.jp/english), is a Web browser developed for

PocketPC and PocketPC2002 devices.
- Opera (www.opera.com/products/smartphone) offers browsers for a large variety of smartphones and PDAs.
- SoftSource (0time.com) is a vector graphic display system and Web browser for handheld systems

Acknowledgments

Dik Lun Lee's work was supported partly by grants from the Research Grant Council of Hong Kong (Grant HKUST 6079/01E). Jianliang Xu's work was supported partly by grants from the Hong Kong Baptist University (Grant FRG/02-03/II-34).

References

1. D. Raggett. *HTML 3.2 Reference Specification*, W3C, 1997, www.w3c.org/TR/REC-html32.
2. F. G.-Castano et al., "Hierarchical Atomic Navigation for Small Display Devices," *Proc. 10 Int'l World Wide Web Conf.*; www10.org/cdrom/posters/1015.pdf, WWW10 Ltd. 2001.
3. 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*, vol. 9, no. 3, May, 1990, pp. 215–227.
4. R.L. Duchnicky and P.A. Kolers, "Readability of the Text Scrolled on Visual Display Terminals as a Function of Window Size," *Human Factores*, vol. 25, no.6, 1983, pp. 683–692.
5. B. Schneiderman, "User Interface Design and Evaluation for an Electric Encyclopedia," *Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems*, Elsevier Science, 1987, pp. 207–223.
6. M. Jones et al., *Improving Web Interaction on Small Displays*, tech. report, Interaction Design Centre, School of Computing Science, Middlesex Univ. 1997.
7. S.J. Lim and Y.K. Ng, "Constructing Hierarchical Information Structures of Sub-page Level HTML Documents," *Proc. 5 Int'l Conf. Foundation of Data and Organization* (FODO '98), Kluwer Academic Publishers, 1998, pp. 66–75.
8. S.J. Lim and Y.K. Ng, "Extracting Structures of HTML Documents," *Proc. 12 Int'l Conf. Information Networking* (ICOIN12), IEEE CS Press, 1998, pp. 420–426; www.computer.org/proceedings/icoin/7225/72250420abs.htm.
9. K.K. Hoi, D.L. Lee, and J. Xu, "Document Visualization on Small Displays," *Proc. Int'l Conf. Mobile Data Management*, ACM Press, 2003, pp. 262–278.

Dik Lun Lee is a professor in the department of computer science at Hong Kong University of Science and Technology. His research interests include document retrieval and management, search engines, and mobile and pervasive computing. He has a PhD in computer science from the University of Toronto. Contact him at dlee@cs.ust.hk.

Ka Kit Hoi has an MPhil in computer science from the Hong Kong University of Science and Technology. His research interests include information retrieval and mobile computing. Contact him at algerhkk@yahoo.com.hk.

Jianliang Xu is an assistant professor in the department of computer science at Hong Kong Baptist University. His research interests include mobile and pervasive computing, location-aware computing, Internet technologies, and wireless networks. He has a PhD in computer science from Hong Kong University of Science and Technology. Contact him at xujl@comp.hkbu.edu.hk.

Wing Sing Dicky Wong is an Mphil student in the department of computer science at the Hong Kong University of Science and Technology. His research interests include mobile computing, spatial databases, and location-dependent systems. He has a BEng in computer engineering from the Hong Kong University of Science and Technology. Contact him at egwws@cs.ust.hk.

DS Online ISSN: 1541-4922 • Feedback? Send comments to dsonline@computer.org.