Web Page Filtering and Re-Authoring for Mobile Users

TIMOTHY BICKMORE[1], ANDREAS GIRGENSOHN AND JOSEPH W. SULLIVAN

FX Palo Alto Laboratory, 3400 Hillview Avenue, Building 4, Palo Alto, CA 94304, USA
[1]Present address: MIT Media Laboratory, Massachusetts Institute of Technology, MA, USA
Email: andreasg@pal.xerox.com

The Digestor system automatically converts web-based documents designed for desktop viewing into formats appropriate for handheld devices with small display screens, such as Palm-PCs, PDAs, and cellular phones. Digestor employs a heuristic planning algorithm and a set of structural page transformations to produce the 'best' looking document for a given display size. Digestor can also be instructed, via a scripting language, to render portions of documents, thereby avoiding navigation through many screens of information. Two versions of Digestor have been deployed, one that re-authors HTML into HTML for conventional browsers and one that converts HTML into HDML for Phone.com's micro-browsers. Digestor provides a crucial technology for rapidly accessing, scanning and processing information from arbitrary web-based documents from any location reachable by wired or unwired communication.

Received August 18, 1998; revised April 21, 1999

1. INTRODUCTION

Several research projects [l, 2, 3, 4] have demonstrated access to world-wide web (WWW) documents from personal electronic devices. Now such access is a commercial reality. Personal digital assistants (PDAs) with networking functionality now come packaged with web browsers, and web-like browsers are commercially available for cellular phones and two-way pagers.

Unfortunately for the mobile users, most web documents are designed for display on desktop computers with color monitors having at least 640 x 480 resolution and many documents are designed with resolutions of 1024 x 768 and larger in mind. This leads to a ratio from four- to 100-to-one of designed versus available screen area, making direct presentation of documents on small devices aesthetically unpleasant, un-navigable and, in the worst case, completely undecipherable. This presents the central problem addressed in our work: how to display arbitrary HTML documents that have been designed for desktop systems on personal electronic devices that have more limited input/output (I/O) capabilities?

In the larger context of mobile and ubiquitous computing, Digestor provides a key technology for giving users view-mobility over platforms [5]. Today, technologies already provide computational mobility and wireless connectivity, but the standard solutions to viewing documents and web pages on tiny screens are to either increase the screen resolution, which is great if you carry a magnifying glass with you, or to provide the ability to fax or print to a local hardcopy device, which is both inconvenient and contradicts the rationale for having electronic documents in the first place. Digestor performs an automatic re-authoring of web documents to make them fit on small displays.

2. PREVIOUS WORK

We have categorized techniques for displaying WWW pages on small screen devices into five general approaches: device-specific authoring, multiple-device authoring, client-side navigation, automatic re-authoring, and web page filtering. This section describes these approaches and analyzes their advantages and disadvantages.

2.1. Device-specific authoring

Device-specific authoring involves authoring a set of WWW pages for a particular display device, for example a cellular phone fitted with a display and communications software such as the Nokia 9000. The basic philosophy in this approach is that users of such specialty devices will only have access to a select set of services, and the pages for these services can all be designed up-front for the device's particular display. Information may be provided from the WWW at large, but the desired pages must be pre-defined and custom information extraction and page formatting software must be written to deliver the information to the small device. This is the approach taken in Phone.com's (formerly Unwired Planet) UPLink service [6] that uses a proprietary mark-up language (HDML).

2.2. Multiple-device authoring

In multiple-device authoring, a range of target devices are identified and mappings from a single source document to a

set of rendered documents are defined to cover the devices within the range. One example of this is the StretchText approach [7], in which portions of the document (potentially down to the word level) can be tagged with a 'level of abstraction' measure. Upon receiving the document, users can specify the level of abstraction they wish to view and are presented with the corresponding detail. Another example of multiple-device authoring is HTML cascading style sheets (CSS) [8]. In CSS, a single style sheet defines a set of display attributes for different structural portions of a document (e.g. all top-level section headings are to be displayed in red 18-point Times font). A series of style sheets may be attached to a document, each with a weight describing its desirability to the document's author. The user can also specify a style sheet, as can the WWW browser using the 'default' style sheet. Although the author's style sheets normally override the user's, the user can selectively enable or disable the author's, providing them with the ability to tailor the rendering of the document to their particular display.

2.3. Client-side navigation

In client-side navigation, the user is given the ability to interactively navigate a single web page by altering the portion of it that is displayed at any given time. Scroll bars on the document display area are one example. Another example is the PAD++ system [9], that lets the user zoom and pan the device display over the document. Active outlining [10] has also been implemented as a client-side navigation technique, in which the user can dynamically expand and collapse sections of the document under their respective section headings. Other techniques that fall into this category include semi-transparent widgets [11] and the Magic Lens system [12].

2.4. Automatic re-authoring

Automatic re-authoring involves developing software that can take an arbitrary web document designed for the desktop, along with the characteristics of the target display device, and re-author the document through a series of transformations so that it can be appropriately displayed on the device. This process can be performed either on the client, on the server or on an intermediary HTTP proxy server (as in [13]) that exists solely for the purpose of providing these transformation services. An example of this latter approach is the UC Berkeley Pythia proxy [14], that performs transformations on web page images, although the focus of this work was on minimizing page retrieval time and not on producing the most appropriate page layout for the display device. More recent work by this group has led to the development of a proxy-assisted web browser for the Palm Pilot ('Top Gun Wingman' [15]). While their system formats HTML pages into layouts appropriate for the client, using only supported text styles and image formats, it apparently does not perform structural transformations such as partitioning pages, building index pages or moving or re-formatting HTML elements. Finally, Spyglass Prism [16] is a commercial product that performs automatic re-authoring of HTML documents using fixed transformations associated with page tags or embedded object types (e.g. reducing all JPEG images by 50%).

2.5. Page filtering

Finally, web page filtering lets users see only those portions of a page they are interested in. The filtering may be performed on an HTTP proxy server to conserve wireless bandwidth and device memory, but it could also be performed on the client as a display-management technique. Filter specifications can be based on keyword or regular expression matching or page structure navigation and extraction commands, and can be either specified using visual tools or a scripting language.

2.6. Trade-offs

Each of these approaches has benefits and drawbacks. Device-specific authoring will typically yield the best-looking results because of the direct involvement of human designers, but limits the user's access to a small select set of the pages available on the web. Multiple-device authoring, while less effort per document than device-specific authoring, still requires significantly more manual design work than simply authoring for a single desktop platform. Client-side navigation can work well if a good set of viewing techniques can be developed, but requires that the entire document be delivered to the client device at once which may waste valuable wireless bandwidth and memory.

Automatic re-authoring does not require any extra work on the part of the web page designers so that all pages are accessible. The re-authored pages require less bandwidth and memory than the complete document. If automatic re-authoring can be made to produce legible, navigable and aesthetically pleasing re-authored documents without loss of information, it is superior to the other approaches. Therefore, we chose automatic re-authoring and worked on designing a good transformation process.

Our automatic re-authoring technique copes well with displays found in PDAs, but, when applied to the very limited displays found on current cellular phones, it sometimes produces pages that are difficult to navigate. We believe that when accessing the web from a cellular phone, most users are mainly interested in accessing very specific information. The filter technique provides them with manual control in defining the information they would like displayed. Page filtering returns only a small portion of a page that is easily navigable. Page filtering is ideal in those situations in which the user is monitoring a particular page whose layout is fixed, but whose content is changing, since they can tune their filter to the format of the page. Conversely, applying filters to pages the user has never seen before may not yield the desired results.

3. THE DIGESTOR SYSTEM

The Digestor system provides an automatic re-authoring capability coupled with page filtering to provide access to

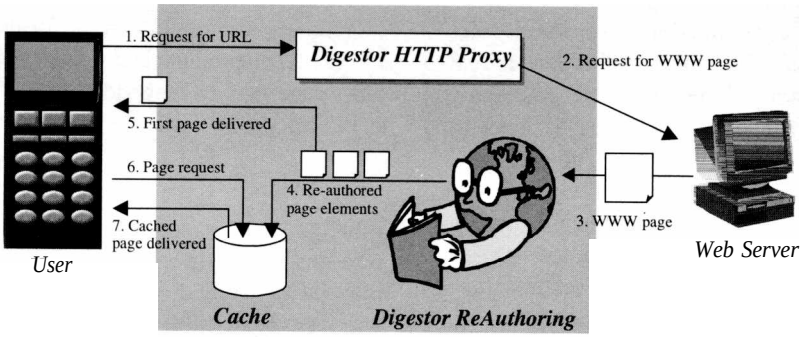

FIGURE **1.** Document flow in Digestor between the user and server.

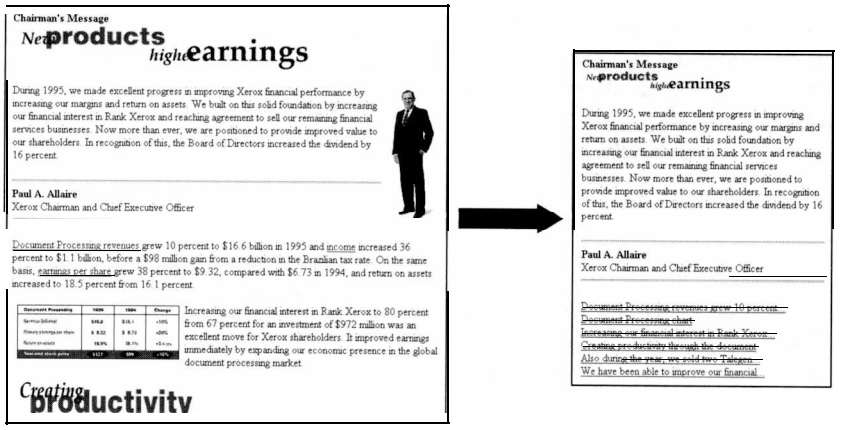

FIGURE 2. A manually re-authored page.

arbitrary documents on the WWW to devices with limited communications bandwidth and small displays.

3.1. Overview

Digestor intercepts requests for web pages and returns re-authored versions rather than the original pages. The first thing that users of Digestor will typically do is specify the size of display for their device and indicate the size of their default browser font; these are required in order to estimate the screen area requirements of the text blocks. Once users have configured the system, they can start retrieving documents from the web. Figure 1 shows the flow of documents among the user, Digestor and the web server. Re-authored documents (each usually partitioned into many smaller pages) are cached to improve efficiency.

Figure 2 demonstrates how a web page can be re-authored for a smaller display. Some of the images have been scaled down and others have been replaced by links. Some text has also been replaced by links. If the user follows any of the generated links, re-authored versions of the replaced passages are automatically returned.

Digestor also supports cellular phones that have very small text displays. Many cellular phones cannot display images. They also do not support links embedded in the text. Instead, they provide programmable buttons that can be used for navigation. Figure 3 illustrates Digestor's re-authoring capability for a cellular phone display. In this example, the front page of the *Wall Street Journal* provides access to several articles. Digestor presents the header information on the first page. The 'Next' button provides access to a menu containing the first few words of each paragraph for direct access to each paragraph. The selection of the first menu item displays the corresponding paragraph. The '[1]' in the paragraph indicates a link. Following the link leads to a page containing the first portion of the article page. The 'Next' button on that page leads to the start of the article.

3.2. Initial study of manual re-authoring

In order to gain an understanding of the process required of an automated re-authoring system, a study was conducted to assess the characteristics of typical web pages and to identify candidate re-authoring techniques through the process of re-authoring several web pages by hand.

A collection of 'typical' web pages-the Xerox Corporate web site [17]—was initially selected as a focus for the study. That collection of web pages was representative of a state-of-the-art, professionally-designed site. A variety of statistics were collected on these pages using a web crawler, to help gain an understanding of the structure and content of a typical page. These statistics generally agree with other, larger-scale studies that have been performed across the entire web [18, 19].

Next, a subset of the pages in the Xerox web site was selected for manual re-authoring. A set of pages from the

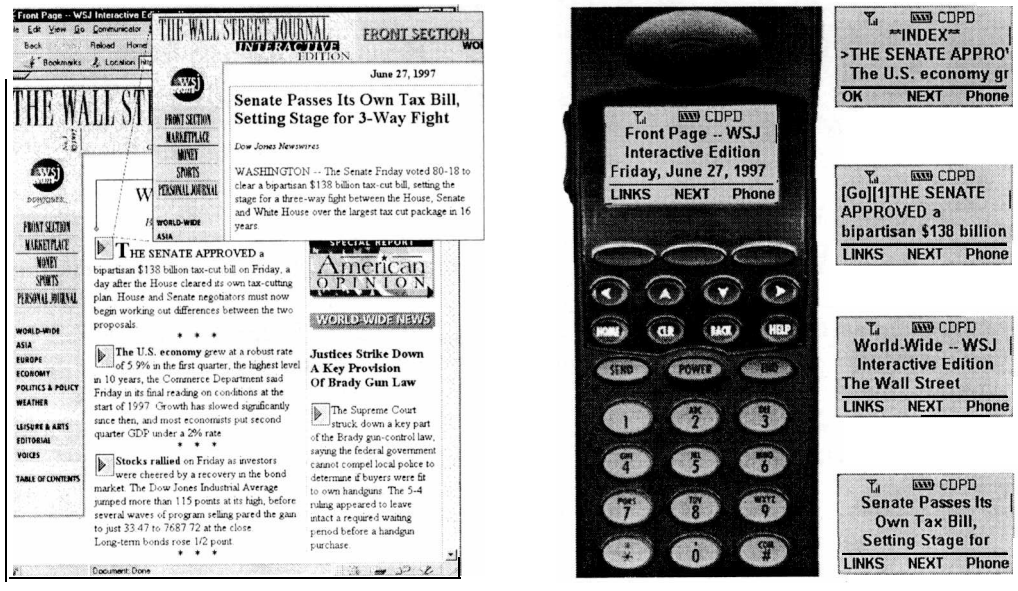

FIGURE 3. Re-authoring of web pages for a cellular phone display.

Xerox 1995 Annual Report [20] were selected and converted by hand for display on a Sharp Zaurus PDA with a 320 x 240 pixel screen (see Figure 2). Detailed notes were kept of the design strategies and techniques used.

Some of the design heuristics learned during this process are as follows.

- Keeping at least some of the original images is important to maintain the look and feel of the original document. Common techniques include keeping only the first, or only the first and last image (bookend images) and eliding the rest.
- Section headers (Hl-H6 tags) are not often used correctly. They are more frequently used to achieve a particular font size and style (e.g. bold), if they are used at all. Thus, they cannot be relied upon to provide a structural outline for most documents. Instead, documents with many text blocks can be reduced by replacing each text block with the first sentence or phrase of each block *(first sentence elision).*
- A rule of thumb for images is to reduce them all in size by a fixed percentage, dictated by the ratio of the display area that the document was authored for to the display area of the target device. Images which contain text or numbers can only be reduced by a small amount before their contents become illegible.
- Semantic elision can be performed on sidebars that present information that is tangential to the main concepts presented in a page. Many of the Xerox pages had such sidebars that were simply eliminated in the reduced versions.
- Semantic elision can also be performed on many images that do not contribute any information to the page, but serve only to improve its aesthetics.
- Pages can be categorized and then re-authored based on their category. Two examples of these are *banners* and *link tables.* Banners primarily contain a set of images and a small number of navigation links (often only one) that serve to establish an aesthetic look, but contain little or no content. Link table pages consist primarily of a set of hypertext links to other pages and very little additional content. These pages can usually be re-formatted into a more compact form that just lists the links in a text block.
- Several techniques were discovered for reducing the amount of white space in a page. Sequences of paragraphs (P tags) or breaks (BR tags) can be collapsed into one. Lists (UL, OL and DL tags) take up valuable horizontal space with their indenting and bullets and can be re-formatted into simple text blocks with breaks between successive items (also observed in [7]).

In conclusion, for document re-authoring two things are required: a set of re-authoring techniques (page transformations) and a strategy for applying them. Of the techniques used in the manual re-authoring study those that were the easiest to formalize were the syntactic elision techniques (section outlining, first sentence elision and image elision) and the syntactic transformation techniques (image size reduction and font size reduction). The design strategy applied during the study consisted of a ranking of the transformation techniques (i.e. *try this before that)* and a set of conditions under which each transformation or combination of transformations should be applied. These strategies influenced the automatic re-authoring techniques described in the next section.

Since the initial study the formatting of web pages has evolved. To determine new transformations and to fine-tune existing ones, Digestor has been tested constantly against a large selection of pages from on-line newspapers and corporate sites. In addition to indicating the need for

TABLE 1. Dimensions of re-authoring techniques.

	Elide	Transform
Syntactic	Section outlining	Image reduction
Semantic	Removing 'irrelevant' content	Text summarization

additional transformations, these pages also demonstrate the many different syntactically incorrect uses of HTML. Our goal has been to handle these syntax errors in the same forgiving manner as done by the standard web browsers.

3.3. Automatic re-authoring in Digestor

The results of the study discussed above led to two major elements to Digestor's design: a collection of individual re-authoring techniques that transform documents in various ways and an automated re-authoring system that implements a design strategy by selecting the best combination of techniques for a given document/display size pair.

3.3.1. Re-authoring techniques

There are many possible automatic re-authoring techniques, that can be categorized along two dimensions: syntactic versus semantic and transformation versus elision. Syntactic techniques operate on the structure of the page, while semantic techniques rely on some understanding of the content. Elision techniques basically remove some information, leaving everything else untouched, while transformation techniques involve modifying some aspect of the page's presentation or content. Table 1 illustrates these dimensions, along with examples of each category.

Currently, Digestor applies only syntactic transformations. Such transformations are less error-prone and more generally applicable. Nevertheless, we are addressing the removal of irrelevant content by offering page filtering and we have started to explore the use of text summarization. This section describes some of the transformations used by Digestor.

Outlining transforms. Section header outlining techniques provide a very good method for reducing the required display size for structured documents, such as technical papers and reports. The outlining process is depicted in Figure 4. The contents of each section are removed from the document and the section header is converted into a hypertext link which, when selected, loads the additional content into the browser. When confronted with multiple section levels (sections, sub-sections, sub-sub-sections, etc.), there are two approaches to performing the elision. The first-fill *outlining-works* by keeping only the section headers and eliding all content, with the results looking like a table of contents for a book. In the second approach-to-level *outlining-a* cut-off level in the section hierarchy is determined and all of the content below that level (including lower-level section headers) is elided, but all of the content above that level is kept.

First sentence elision transform. Since most pages have text blocks, even when no section headers are present, first sentence elision can be a good way of reducing the required screen area. In this technique, each text block is replaced with its first sentence (or phrase up to some natural break point), and this sentence is also made into a hypertext link to the original text block.

Indexed segment transform. This transform takes an input page, segments the content into sub-pages by allocating some number of items to each and builds and prepends an index page to the collection. Indexed segment transform first attempts to find page elements that can be logically partitioned, such as ordered or unordered lists, sequences of paragraphs or tables, and starts filling output pages with these elements in order until each is 'full' with respect to the client's display size. If a single logical element cannot fit on an output page then a secondary partitioning is performed that partitions text blocks on paragraph or sentence boundaries. As much style information as possible is retained for the output elements by outputting each element embedded within all of its ancestor's tags in the tree. The index page is constructed by copying a section header or first sentence from each element output, concatenating them onto the index page and creating a hypertext link from each to the appropriate sub-page. The index page itself may need to be segmented. 'Next' and 'Previous' navigation links between sequential sub-pages are also added for navigational convenience.

Table transform. The table transform recognizes when a table (presentation of information arranged in a rectangular grid) on the input page cannot be directly sent to the client. In these cases, it outputs one sub-page per table cell, in top-down, left-to-right order. Tables nested within tables are processed in the same manner. The table transform uses heuristics to determine when table columns are being used as 'navigational sidebars' (common practice in commercial HTML web pages) and moves these cells to the end of the list of sub-pages since they tend to carry very little content. Figure 5 shows a nested table, marking tables with thicker borders than table cells. Cell 1 is identified as sidebar and will be placed after cell 6. All the other cells are placed in their natural order. The portions of cell 4, 4a and 4b, are each placed on their own sub-page unless they contain only white space.

As one can see from the example, nested tables and sidebars complicate the processing of tables. This is especially true if sidebars are part of an inner table. In that situation, the sidebar should move to the end of the inner table, but not to the end of any surrounding tables. Digestor's approach is to move sidebars one table at a time and then to process all table cells at once rather then grouping them by table.

Image reduction and elision transforms. Images present one of the most difficult problems for automatic re-authoring, because the decision of whether to keep, reduce or eliminate a given image should be based on an understanding of the

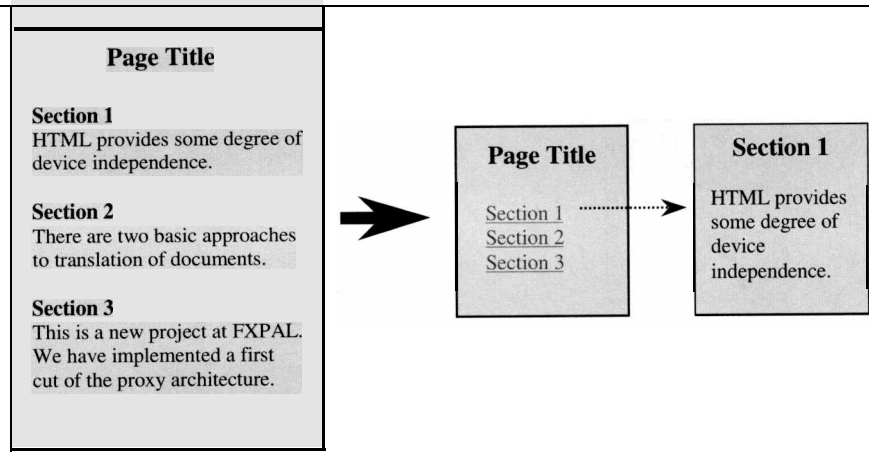

FIGURE 4. Section outlining transform.

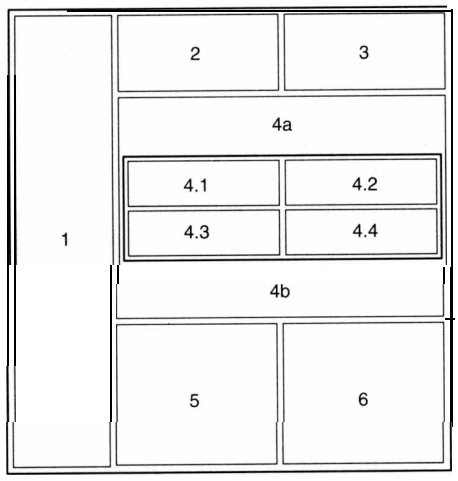

FIGURE 5. Nested table.

content and the role of the image on the page. However, image reduction and elision can be applied without content understanding, as long as users are provided with a mechanism by which they can retrieve the original image. The approach taken in Digestor is to provide a set of techniques that transform all images in a page by pre-defined scaling factors (25%, 50% and 75%) and making the reduced images hypertext links back to the originals. In addition to image reduction, three syntactic elision techniques were also developed for images--elide all, first image only and bookends-in which all images, all but the first and all but the first and last are elided from the document, respectively. Elided images are replaced with their ALT text when available or with a standard icon when no ALT text is available, which is also made into a hypertext link to the original image.

Image map transform. If screen space is too limited or the client device cannot display images, Digestor will remove them from the document. However, images can be used as anchors for hypertext links via a client-side image map. If such images are removed, the web site can be rendered non-navigable. To accommodate this, Digestor incorporates a transform that extracts the hypertext links from such images and formats them into a text list of link anchors. The labels for the text list are extracted from the ALT tags of the image map, if present, or from part of the URL of the link. This transformation preserves links attached to images for navigation when removing the images.

3.3.2. Automated re-authoring system

In this section, we describe the re-authoring engine that uses heuristics to generate pages customized for the specific device upon which they will be displayed. Individual page transformations are ordered by their desirability. In order to determine which combination of transformations should be applied to a given document Digestor performs a depth-first search of the document transformation space, using many heuristics that describe preconditions for transformations and combinations of transformations. The depth-first search ensures that a 'good enough' version of the document is found by using a combination of the most desirable transformations. Only if the more desirable transformations are not applicable or do not reduce the document enough are the less favored transformations used. Each state in the transformation space represents a version of the document, with the initial state representing the original 'as-authored' document (see Figure 6). Each state is tagged with a number that represents the quality of the document version represented. A state can be expanded into a successor state through the application of a single transformation. As soon as a state is created containing a document version that is good enough, the search is halted and that document is returned to the client for rendering. If the search is exhausted and no document version can be found that is good enough, then the best document found during the search is returned. If there are hard size constraints that are not met by the best document, a more destructive transformation is applied that breaks the documents up in the middle of paragraphs.

Heuristic information is used in several places in the planner, such as the order in which transformation techniques are applied to a given state, the preconditions for each transformation technique and the determination of when a document version is good enough to halt the search. In general, transformations that make minor changes to the document are preferred over those that make

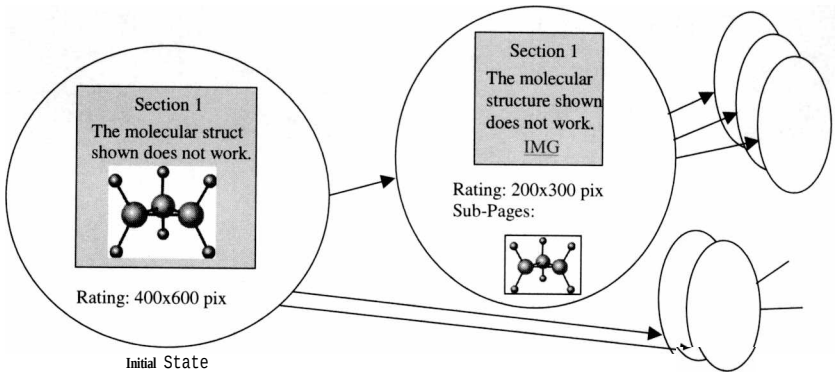

FIGURE 6. The document version search space.

more extensive changes (e.g. reducing images by 25% is preferable to reducing them by 75%). The preconditions for each transformation technique specify which other transformations they can be combined with (e.g. it makes no sense to apply both full outlining and first sentence elision to the same document), as well as requirements on the content and structure of the document that the technique is being applied to (e.g. only apply full outlining when there are at least three section headers). The current condition for 'good enough' is fairly simplistic; the search is stopped when the area required by a document version is some multiple (typically 2-5) times the screen area of the client display, which assumes that the user does not mind scrolling the display a little in one direction.

When a transformation is applied to a document, it can result in the document's contents being split into multiple, smaller 'sub-pages' (as shown in Figure 4). However, each of these sub-pages may still be too large to download and display on the client. To address this problem, Digestor keeps a list of the sub-pages generated by each sequence of transformations attached to the state representing the resulting document version (see Figure 6). Once the good enough version of the document is selected (that is really only a good enough version of the first sub-page delivered to the client), the list of generated sub-pages for that version is added to a global list of pages to be re-authored. Digestor then re-authors each of these to-be-re-authored pages until all of the resulting sub-pages can be delivered to the client. This algorithm is shown in pseudo code below, where 'reauthor' refers to the best-first re-authoring process described above for a single input page.

```
Digestor(initial_page)
   to_be_reauthored = { initial_page }
   to-deliver = {}
   while(to_be_reauthored != {})
      next_page = pop(to_be_reauthored)
      best_version_state = reauthor(next_page)
      to_deliver.append(best_version_state.page)
      to_be_reauthored.append(best_version_state.sub_pages)
  return to-deliver
```

All re-authored sub-pages are cached by Digestor as transformed parse trees. As the user navigates a transformed document and requests sub-pages, the corresponding trees are rendered in a markup language and sent to the client.

3.3.3. Implementation of page re-authoring

Digestor re-authors documents by first parsing them and constructing a parse tree or abstract syntax tree (AST) representation, applying a series of transformations to the tree and then mapping the tree back into a document representation (that may be in a document format that is different from the input format). The complexity lies in which subset of the many possible transformations should be applied.

Document transforms are implemented using a standard interface that consists of a condition function that takes a state node in the document version space and returns true if the transform can be applied to the state, and an action function that is called by Digestor when the transform is actually applied to a state to produce a new state containing a new document version, measure of quality and resulting sub-pages. Three types of transforms can be defined: (1) those which are always run on a page before the planning process starts, (2) those used in the best-first planner and (3) those which are always run on a page before it is translated from the final AST back into a surface form such as HTML.

Transformations manipulate the AST in the state they are applied to in order to produce a new version of the document (the manipulations are similar to those described in [21]). Whenever portions of the AST are elided or transformed an HTML hypertext link is added into the AST referencing the node identifiers of all of the affected AST subtrees, enabling users to request the original portions of the document that have been modified during re-authoring.

Digestor also keeps track of which combinations of transforms have already been tried, via a global list of transform sets (assuming that all transformations are commutative), to ensure that no duplicate states are ever constructed.

The design described above has been implemented as an HTTP proxy server (as in [13]). The proxy accepts a request for an HTML document, retrieves the document from the specified HTTP server, parses the HTML and constructs an AST, labels each of the AST nodes with a unique identifier and then retrieves any embedded images so that their size can be determined (as necessary). The system is implemented in the Java programming language

(approximately 10,000 lines of code) and the HTTP proxy server software was initially based on the MBServler system [22] before we made extensive changes to deal with proxy server chains and to implement our own caching scheme. In addition to functioning as a true proxy, this system can also respond to requests for certain URLs with documents generated by the proxy itself, that is used to provide the user with forms-based control over the proxy and there-authoring process.

3.3.4. Handheld device markup language

The initial version of Digestor was designed to support a range of PDA class devices that could display HTML [23]. Subsequently, we created a new version aimed at even smaller devices such as cellular phones with a character display. Several of these cellular phones support the handheld device markup language (HDML). Cellular phones do not have a pointing device so that the keypad has to be used for navigation. HDML 2.0 does not support images or embedded links so that alternative means have to be used to present navigation choices. Only one form field can be presented on a page so that HTML forms have to be transformed into a sequence of HDML pages. To address these differences Digestor uses a number of transforms specific to HDML. All transformations still take place by manipulating an HTML parse tree. Different rendering modules produce HTML or HDML from the transformed tree as a final processing step (as in the TACC system described in [15]). Our experience in using this version of Digestor led to the implementation of filtering because the very small size of the cellular phone display sometimes caused the generation of many sub-pages that were difficult to navigate.

3.4. Web page filtering

Even with perfect automatic re-authoring of web documents, there is simply too much information in a typical web site to make serendipitous cellular phone web browsing a pleasurable or profitable past-time. Typically, these devices and services will be used to find and present information that the user is specifically looking for; that is, for targeted information search and extraction. Digestor allows users to extract only the portions of documents that they are interested in, via a simple, end-user scripting language that combines structural page navigation commands with regular expression pattern matching and report generation functions.

3.4.1. Related work in page filtering

The SPHINX system [24] provides a visual tool that lets users create custom 'personal' web crawlers, that are similar in functionality to Digestor's filtering mechanism. The Internet Scrapbook [25] allows users to visually select elements from web pages and then updates these elements in a 'scrapbook' when the web pages change, providing a function that is similar to Digestor's page element retrieval for a particular page. Several commercial products also provide similar functionality for other applications (e.g. corporate reporting or database population). Lanacom's Headliner Pro [26] and OnDisplay's CenterStage [27] both provide visual editors that let users specify which structural parts of web pages to extract (noticeably absent is any ability to extract content based on regular expressions or keywords). Agentsoft's LiveAgent Pro [28] and WebMethods Web Automation Toolkit [29] both include scripting languages that provide the same ability to designate portions of web pages as Digestor. They require programmer-level knowledge to use, whereas Digestor's scripting language is geared more towards end users (e.g. it has only four basic commands). Finally, the World-Wide Web Consortium's Document Object Model [30] specification defines a mechanism for navigating the elements of a page from within a browser program or script (e.g. Java or JavaScript). This specification has been partially implemented in Microsoft's Dynamic HTML capability [3l] included in their Internet Explorer 4.0 browser. However, this approach requires programming expertise to utilize and currently does not work in server or proxy environments, thus using it would require downloading the entire document to the client, which may be costly or infeasible. None of these systems or products allow extracted content to be re-authored for small display devices.

The notion of page filtering is related to the concept of aggregation as described in [15], in which the proxy can search multiple web sites for specified information and combine the results before passing the information back to the client. However, the aggregation processes used in this system are programmer-developed modules and thus do not support end-user authorable filters.

3.4.2. Digestor's approach to page filtering

Digestor has the capability to extract partial information from a document based on commands written by a user in a high-level scripting language. This filtering module combines page structure navigation, regular expression matching, site traversal (web crawling) and iterative matching, in addition to re-authoring of the extracted information using the techniques described above. A filter script is simply entered into a text file and saved on a web server; it is executed whenever a user requests its URL (assuming they are using Digestor as their HTTP proxy). A filter script will typically load a target web page, traverse to particular locations within the page (described structurally and/or by regular expressions), extract the content found at those locations and then send it through the re-authoring system to be properly formatted before being returned to the user (see Figure 7).

3.4.3. Navigation within a web page

The filtering module takes advantage of the parse tree creation and navigation routines in Digestor by providing a simple set of web page navigation options that **use** the concept of a 'current context' in the web page. The current context is analogous to a 'cursor' in database programming, in that it refers to a location within the document. In

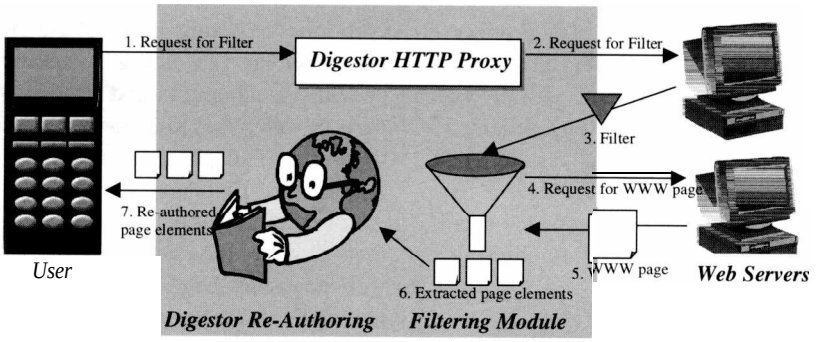

FIGURE 7. Example of dataflow in the document filtering module.

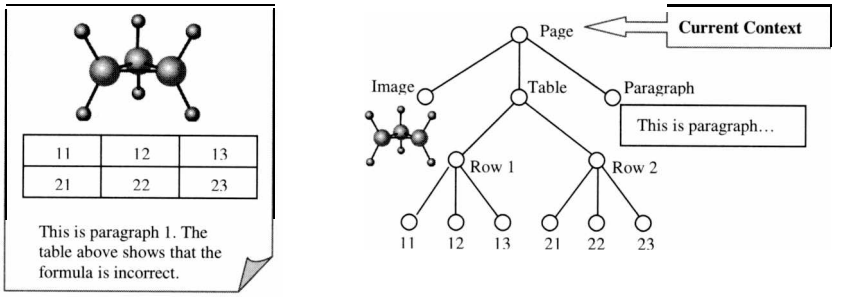

FIGURE 8. Example WWW page and corresponding AST. The current context is set to root of the AST

actuality, the current context refers to a node in the HTML parse tree and the navigation commands serve to move this reference around within the tree until a desired part of the page is found, at which time it can be extracted. For example, Figure 8 shows a web page and its corresponding AST. When the page is first loaded into the filtering module (by executing a **GO URL** command) the current context is pointing at the root node of the AST, that essentially refers to the entire document.

There are three forms of page navigation commands, those which go *into* the current context (selecting more specific content), those which go *out* from the current context (to enclosing structures) and those which traverse the page sequentially from the start of the current context (e.g. to navigate to the *next* structure of some kind, that may or may not be properly contained within the current context).

The simplest form of navigation command goes *into* the current context. For example, given the document and current context shown in Figure 8, executing GO **ROW 2** results in the current context being moved to the second table row object within the current context, as shown in Figure 9.

The current context can also be enlarged (i.e. moved up the AST towards the root) by using a **GO ENCLOSING** command. For example, given the document and context shown in Figure **9,** a **GO ENCLOSING TABLE** command results in the current context shown in Figure 10.

Finally, the current context can be moved forwards or backwards among the objects in a page in a sequential manner (as they appear to a reader). This is accomplished by moving the current context forwards or backwards from its current location within a prefix traversal of the AST. This results in a search that first is performed within the current context, then continues with the objects that follow the current context on the page. For example, **GO PREVIOUS IMAGE** moves to the previous image found sequentially from the current context.

In addition to named page elements, navigation commands can also be specified using regular expressions. For example, **GO NEXT "DOW\sJONES\s*(\d+)\s*POINTS"** moves the current context to the next match of the specified regular expression (using a prefix traversal of text blocks on the page). The current implementation of Digestor supports the Per15 regular expression syntax [32], including the ability to demarcate sub-expressions and recall them into output strings.

The implementation of this navigation scheme is a straightforward manipulation of a current context structure to point at one of the nodes in the AST. Two areas in which this technique does not work directly are in the handling of regular expression searches and table navigation. When a regular expression search is commanded, the filter processes the text blocks of the document in sequence looking for a match within each. When a match is found it is marked by setting the current context to the text block node and annotating it with the fact that the match is actually on a substring of the node and the start and end indices of the substring within the text block. In table processing, table, row and cell navigation are straightforward (assuming the HTML model of a table consisting of rows each of which consists of cells), however column navigation

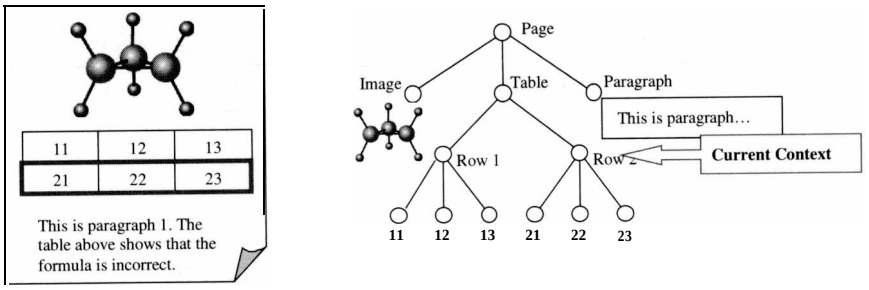

FIGURE 9. Example of current context after GO **ROW 2** is executed.

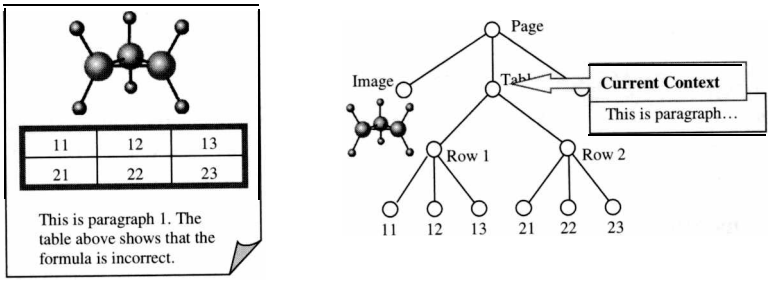

FIGURE 10. Example of current context after GO **ENCLOSING TABLE** is executed.

requires special treatment. When the user issues a table column navigation command (e.g. GO **SECOND COLUMN**) the current context is set to the table node and annotated with the fact that it is really pointing at a column within the table and the column number.

3.4.4. Navigation between web pages

The simple navigation commands described above can also be used to navigate among a set of linked web pages through the use of the **LINKEDPAGE** page object type. For example, **GO FIRST LINKEDPAGE** moves to the first hypertext link within the current context, loads the referenced page and moves the current context to the root of its AST, while GO **ENCLOSING LINKEDPAGE** returns the current context to the hypertext link that led to the page currently being processed (swapping a previously loaded page back in for processing).

Traversal between pages is handled by a stack of script activations, each of which pairs script state information (including current context) with a particular URL and AST. This facilitates rapid navigation back and forth among linked pages and is required to support the **GO ENCLOSING LINKEDPAGE** command.

3.4.5. Filter output

Once the current context has been moved to a page object that is of interest, the **REPORT** command is used to extract it. **REPORT** can be issued several times within a filter, in which case the extracted page elements are concatenated before being run through Digestor's re-authoring module. **REPORT** can also be used to insert arbitrary strings into the output, that can contain sub-strings from regular expression pattern matching. For example **REPORT "Dow:\1"** adds the string 'Dow:' plus a substring extracted during a regular expression match to the filter's output.

3.4.6. Repetitive search

Often one does not know in advance how many page elements of a particular kind will exist on a web page (e.g. news article paragraphs in a daily e-zine). The **FOREACH** command addresses this problem by executing a sequence of commands for every page element found within the current context that meets a specified criterion. When used with a **LINKEDPAGE** target, this provides the functionality of a web spider that can visit all of the linked pages within a web site. (Specification of the spider search depth and other criteria are specified via a **PRAGMA** options command.) In the following examples the ellipses represent sequences of valid filter commands:

FOREACH PARAGRAPH DO.. .END	Moves to each paragraph within the current context in turn and executes the specified commands.
FOREACH LINKEDPAGE DO.. .END	Loads each page that is reachable through hypertext links from the current page in turn and executes the specified commands.

Whenever a filter encounters any kind of error, including navigation failures, regular expression matching failures or web page retrieval error, it simply begins the next iteration of the innermost **FOREACH** loop in which the offending command is embedded. If the error occurred at the top level of a filter, the filter halts execution and produces any pending output.

TABLE 2. Optimizations.

Solaris 2.6 on a Sun SPARCstation 20	Transform time (s)
Initial version	26.8
JDK 1.0.2 → JDK 1.1.3/1.1.5	20.3
Remove hash tables, better check for special characters	12.6
Increase initial heap from 1 MB to 5 MB	11.5
Remove all debug messages	7.9
Replace String.toUpperCase, strings instead of URLs	6.9
Replace vectors with arrays	4.9
Use Sun JIT compiler	2.8
Use non-native thread implementation ('green' threads)	2.0
JDK 1.1.3 → JDK 1.2b2 (native/green)	2.1/1.9

4. PRACTICAL ISSUES

4.1. Debugging

Debugging a system such as Digestor is a difficult endeavor, especially considering the magnitude of pages containing syntactically incorrect HTML. Furthermore, for small devices some pages are transformed into many small pages that all need to be visited to verify the correctness of the transformation. To tackle this problem, we created a collection of debugging tools. Some problems can be found automatically, most notably the case in which a transform creates a page without providing a link to it. To get good coverage for such problems and the even more severe problems of Java exceptions, we tested Digestor with 7000 different URLs to HTML pages that we extracted from the log of our lab's proxy server. About 2% of the pages had problems that we traced back to a handful of bugs in the transforms. Other debugging tools enable us to display the parse tree or the source of the complete set of transformed pages. These displays can be requested by adding optional parameters to a URL requested from Digestor. Other optional parameters clear the cache, turn on debug messages and time different modules. The ease of requesting this additional information proved to be invaluable in the development of Digestor.

4.2. Optimizations

For good user acceptance and scalability in a shared server environment Digestor needs to respond reasonably quickly to a request. We established the requirement that most pages should be returned within approximately three seconds. This was no problem for subsequent pages created by a transform because all of the planning took place initially and the cached transformed tree only had to be rendered in a markup language. However, the initial page took significantly longer. As our test page, we used the title page of the *Wall Street Journal* that had a mixture of nested tables, images and other layout elements. On a Sun SPARCstation 20, it took almost 27 s to transform this page.

At that point, we had the choice to reimplement the whole system in C++ or to find ways to speed everything up by a factor of ten. A Windows Native executable created by Symantec Visual Cafe or even Symantec's JIT compiler under Windows would have accomplished the required speed-up, but our target platform was a Sun workstation. Using different versions of the Java Developer Kit (JDK) and a Sun JIT compiler provided significant speed-ups, but they were nowhere as dramatic as the ones provided by the Symantec tools and in the end, insufficient. Our experiences with optimizing the Java code might be informative for other Java developers. Table 2 shows the effects of applying different optimizations and using different Java virtual machines.

The Java profiler was instrumental in determining the bottlenecks in the code. There were three instances in which Java provided primitives for elegantly representing data that turned out to be inefficient. In the first instance, we used hash tables to represent attribute-value pairs attached to each node in the parse tree. Because there were relatively few attributes for each node, hash tables did not provide any advantage in accessing attributes, but caused slow-downs whenever a parse tree was copied during a transform. Representing attribute-value pairs as object arrays that were searched linearly increased the performance significantly. Vectors provided a convenient means for representing the children of a node in the parse tree that supported the addition and deletion of children. Unfortunately, the access methods for a vector are thread-safe that incurs a major performance penalty if many vector elements are accessed that commonly happened during tree traversal. Again, the use of arrays provided a good alternative. Finally, it turned out that print messages, string manipulations and the use of URL objects caused further significant slow-downs. After all of the optimizations, Digestor was 13 times faster; half of which (3.5 times) was due to the use of faster virtual machines.

5. FUTURE WORK

Two versions of the Digestor system have been implemented and tested, and have demonstrated the feasibility of fielding such a service in a wide-scale deployment. While it is possible to use Digestor to browse the web on a cellular

phone or two-way pager, more work-especially in the areas of user testing and navigation design-needs to be performed to perfect the system. Some of the specific areas that we are currently investigating include the following.

5.1. More user control

Users should be able to adjust the various heuristics used in the planner to suit their taste. For example, they could specify the relative preference of the transformation techniques, or specify that some transforms are not used. At a higher level of abstraction, they could express their preferences within a space of trade-offs, such as 'more content' versus 'larger representation'. In addition, the re-authoring system could be moved to the client and coupled with the browser so that the user could dynamically apply and undo different transformations until they achieved a result they liked.

Independently of giving the users more control, we intend to perform user studies to determine the quality of the transformation used by Digestor. The quality of transformations is determined both by the aesthetic appeal of the generated pages and by the ease of navigation. Both aspects can be tested and improved in user studies.

5.2. Improved measure of merit

The current system simply adds up the space requirements of all of the images and text to arrive at an estimate of the screen area requirements for a document. This is adequate for fairly dense documents with minimal structure, such as those in the Xerox Annual Report, but works poorly for documents with a lot of white space or that use advanced layout techniques (e.g., tables). Ultimately, what is needed is a size estimator that performs much of the work performed by a browser in formatting each document version onto a display area. Factors other than the required screen area may also need to be included, such as actual width requirements (users do not like to scroll horizontally), bandwidth requirements and aesthetic measures.

5.3. More transformation techniques

There are several additional transformation techniques that could be explored. Some specific candidates to add include: elision of advanced page elements (e.g. applets, shockwave plug-ins, etc.), folding of table rows or columns and text block summarization via natural language understanding techniques. A thorough study of graphic art techniques could be made to derive additional candidates. Transformations could also be generalized by the use of user-specified parameters so that, for example, a transformation could elide all text blocks containing a user-specified keyword.

5.4. Specification of filter scripts

As simple as the Digestor filtering command language is, we cannot expect end users to program scripts of great complexity. Some of the alternatives that we are investigating include specification of scripts via a visual page navigation tool and the possibility of providing users with a library of pre-built scripts for specific purposes. Another area of investigation is providing users with the ability to specify scripts on their portable device which can then be posted to the Digestor proxy for execution.

5.5. Proxy infrastructure

Finally, several infrastructure design issues need to be addressed before Digestor could be fielded to support a large number of users. These issues include such problems as the proxy server scalability, page de-caching criteria and tracking user preferences and client configurations. One set of solutions to these issues is embodied in the TACC system developed at UC Berkeley [33], which has been in operation supporting over 10,000 Palm Pilot users with an HTML proxy service. This project has focused on these infrastructure issues and thus complements the work done to date on Digestor; our page transformation and filtering techniques could easily be integrated into their framework as additional 'worker' modules.

6. CONCLUSIONS

The increasing ubiquity of mobile devices is opening up new markets for information delivery. Digestor, and other similar approaches, provide a crucial technology for rapidly accessing, scanning and processing information from arbitrary documents from the vast repository known as the world-wide web, from any location reachable by wired or unwired communication. Reading daily financial reports, checking up on competitors or even serendipitous browsing while riding the train into work are all enabled once web documents can be accessed and reviewed from portable wireless devices.

One solution being proffered by companies such as Phone.com (formerly Unwired Planet) requires that all information providers re-author their pages for a particular client device or range of devices. While many businesses will undoubtedly jump at the chance to re-target their services for the wireless market, it is unreasonable to expect that any more than a tiny fraction of the sites on the web will be converted. Digestor provides a bridge that allows users of these specialty services to access arbitrary documents from the web-such as competitors' press releases-that will likely never be converted into these device-specific formats.

Although Digestor is currently designed to work as a proxy server within a document pull model, it could easily be adapted to a range of other possible architectures. Digestor could be used server-side within a document push model to re-author pages before they are pushed to the client (e.g. as email messages). Digestor could also be run directly on the client to provide dynamic re-authoring—allowing the user to interactively modify the re-authoring strategy until an optimal rendering is achieved for the purpose at hand.

Ultimately, the best strategy for providing document access to small portable devices will likely be a collection

of techniques that the user can select from, based on their current needs. For targeted information lookup, technologies such as Digestor's filtering mechanism or other search tools are appropriate. For skimming or gisting, Digestor's approach to building layered document indices or text summarization techniques can be applied. For update monitoring, server-side push or notification technologies are probably best. If the development of the WWW is a relevant guide, there will be a multitude of access and presentation tools and applications developed as these devices become ubiquitous.

ACKNOWLEDGEMENT

Thanks to Bill Schilit who developed the original concept for Digestor.

REFERENCES

[1] Bartlett, J. (1995) *Experience with a Wireless World Wide Web Client.* Technical Note TN-46, March, Digital Western Research Laboratory.

[2] Gessler, S. and Kotulla, A. (1995) PDAs as mobile WWW browsers. *Comput. Networks ISDN Syst.*, **28,** 53–59.

[3] Voelker, G. and Bershad, B. (1994) Mobisaic: an information system for a mobile wireless computing environment. In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, pp. 185-190. IEEE Computer Society Press.

[4] Watson, T. (1994) Application design for wireless computing. In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, pp. 91-94. IEEE Computer Society Press.

[5] Dearl, A. (1998) Towards ubiquitous environments for mobile users. *IEEE Internet Comput.*, **2,** 22–32.

[6] Unwired Planet, Inc. (1996) *UP.Link Developer's Guide Version 1.0.*, Redwood Shores, California. *http://www.phone.com*

[7] Cooper, I. and Shufflebotham, R. (1995) *PDA Web Browsers: implementation issues.* University of Kent at Canterbury Computing Laboratory WWW Page. *http://alethea.ukc.ac.uk/Dept/Computing/ Research/STEP/Papers/WWWPDA/*

[8] Lie, H. and Bos, B. (1996) Cascading style sheets. *WWW Consortium. http://www.w3.org/TR/REC-CSS1*

[9] Bederson, B. and Hollan, J. (1994) Pad++: A zooming graphical interface for exploring alternate interface physics. In *Proc. ACM User Interface Software and Technology '94*, pp. 17-26. ACM Press.

[10] Hsu, J., Johnston, W. and McCarthy, J. (1994) Active outlining for HTML documents: An X-mosaic implementation. In *2nd Int. World Wide Web Conf.*, Chicago, IL. *http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/HCI/hsu/ hsu. html*

[11] Kamba, T., Elson, S., Harpold, T., Stamper, T. and Sukaviriya, P. (1996) Using small screen space more efficiently. In Tauber, M. J., Bellotti, V., Jeffries, R., Ma&inlay, J. D. and Nielson, J. (eds) *Human Factors in Computing Systems [CHI 96]*, Vancouver, Canada, pp. 383-390. ACM Press, New York.

[12] Bier, E., Stone, M., Pier, K., Buxton, W. and DeRose, T. (1993) Toolglass and magic lenses: The see-through interface. In *Proc. 20th Annual Conf. Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 73-80. ACM Press, New York.

[13] Brooks, C., Mazer, M., Meeks, S. and Miller, J. (1996) Application-specific proxy servers as HTTP stream transducers. *World Wide Web J.*, **1,** 539-548. *http://www.w3j.com/1/*

[14] Fox, A. and Brewer, E. (1996) Reducing WWW latency and bandwidth requirements by real-time distillation. *Comput. Networks ISDN Syst., 28*, 1445-1456.

[15] Brewer, E., Fox, A., Goldberg, I., Gribble, S., Lee, D. and Polito, A. (1998) Experience with Top Gun Wingman: A proxy-based graphical web browser for the 3Com PalmPilot. In *IFIP Int. Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pp. 407–424. Springer, Berlin.

[16] Spyglass Software. *http://www.spyglass.com/products/prism/*

[17] Xerox Corporation Inc. (1999) *http://www.xerox.com*

[18] Bray, T. (1996) Measuring the web. *Comput. Networks ISDN Syst.*, **28,** 993-1005.

[19] Woodruff, A., Aoki, P., Brewer, E., Gauthier, P. and Rowe, L. (1996) An investigation of documents from the WWW. *Comput. Networks ISDN Syst.*, **28,** 963-980.

[20] Xerox Corporation Inc. (1995) *Annual Report. http://www.xerox.com/annualreport/1995/*

[21] Bonhomme, S. and Roisin, C. (1996) Interactively restructuring HTML documents. *Comput. Networks ISDN Syst.*, **28,** 1075–1084.

[22] Mort Bay Consulting. *MBServler: Mort Bay's HTTP servlet serving server. http://www.mortbay.com*

[23] Bickmore, T. and Schilit, W. (1997) Digestor: device-independent access to the world wide web. *Computer Networks and ISDN Systems*, 29(8-13), 1075-1082.

[24] Miller, R. and Bharat, K. (1998) SPHINX: A framework for creating personal, site-specific web crawlers. In *7th Int. World- Wide Web Conference*, Brisbane, Australia, pp. 119-130.

[25] Sugiura, A. and Koseki, Y. (1998) Internet Scrapbook: automating web browsing tasks by programming-by-demonstration. *Comput. Networks ISDN Syst.*, **30,** 688-90.

[26] Lanacom, Inc. *http://www.headliner.com*

[27] OnDisplay, Inc. *http://www.ondisplay.com*

[28] Agentsoft, Ltd. *http://www.agentsoft.com*

[29] WebMethods, Inc. *http:/www.webmethods.com*

[30] World-Wide Web Consortium Document Object Model, *http://www.w3.org/DOM/*

[31] Microsoft Corp. Dynamic HTML Specification. *http://msdn.microsoft.com/workshop/author/*

[32] Vromans, J. (1996) *Per15 Desktop Reference.* O'Reilly, Sebastopol, CA.

[33] Fox, A., Gribble, S., Chawathe, Y. and Brewer, E. (1998) Adapting to network and client variation using infrastructural proxies: lessons and perspectives. *IEEE Personal Commun.*, 5, 10-19.