CHAPTER Four

Web Browsing by Example

ATSUSHI SUGIURA

Human Media Research Laboratories, NEC Corporation

Abstract

This chapter describes programming by example (PBE) techniques to automate daily Web browsing tasks. A number of research efforts have tried to solve two major problems of PBE: difficulties in inferring user intent behind examples and low accessibility to application's internal data. However, these problems can be avoided in the Web-browsing domain, and practical PBE systems can be developed. Internet Scrapbook and SmallBrowse are presented here as two examples of such practical systems.

4.1 Introduction

Web browsing is a highly repetitive task. We have many Web pages to visit every day, seeking such information as news, sports results, and weather forecasts. Users of a Web browser are forced to repeatedly perform the same sequence of operations: going to the same Web pages, scrolling to the same positions where users' desired information would appear in those pages, and following the same hyperlinks.

One way to automate Web browsing tasks is to write programs. For example, a user could avoid repetitive accesses and scrolling operations by writing a program for generating personalized pages—that is, dynamically collecting articles that he or she usually browses from multiple pages and merging them into a single page. However, most Web users are non-programmers, and even for expert programmers, it is tedious to make programs that can handle various types of Web documents with different formats.

Programming by example (PBE) is one of the possible solutions to this problem. PBE is a technique that creates an executable program from example data and/or a sequence of example operations given by the user. This PBE framework allows users with little programming skill to automate repetitive tasks without learning any complicated programming languages.

In spite of this great advantage, only a few PBE systems have been of practical use to date. One of the main reasons is that PBE systems cannot always infer the user intent behind the given examples. As long as it is not guaranteed that the system will generate a program as the user desired, no user would make a program with PBE. Another reason is that, in general, a PBE component cannot be attached to existing applications with which the user is familiar. Although many approaches have been taken to these problems (Cypher 1993), one important topic in the discussion was lacking: application domains. Since the degree of difficulty in solving these problems depends on the application domains (text editing, graphical editing, Web browsing, or others), we should further discuss this issue.

This chapter will first explain the general problems of PBE and show how they can be avoided to some extent in the Web browsing domain. Then, two PBE systems for Web browsing will be described *Internet Scrapbook* and *SmallBrowse,* followed by a concluding discussion of their strategies and feasible applications.

Internet Scrapbook (referred to simply as "Scrapbook") automates the repetitive operations in daily Web browsing activities. It allows a user to create a single personal page by copying only the necessary data from multiple Web documents, and it automatically updates itself by extracting the user-specified portions from the latest Web pages. SmallBrowse predicts the hyperlinks that a user would next follow based on that person's browsing history and generates shortcut links to avoid searching a Web page for links. While Scrapbook is designed for Web browsing using desktop computers, SmallBrowse is aimed at facilitating the Web-browsing tasks in handheld computers with small screens.

Scrapbook is one of the first PBE systems that is being marketed commercially. It has been shipped in Japan since July 1997.

4.2 Underlying Problems of PBE

There are two major problems in PBE: how the system can infer user intent behind the given examples and how the PBE component can work with existing application by accessing their internal data.

4.2.1 Problem of Inferring User Intent

PBE systems are required to convert examples/demonstrations into generalized programs as the user desired. This means that systems have to infer the user intent behind these examples/demonstrations. However, such inference systems sometimes guess wrong. The users would not willingly use the PBE systems without guarantees that the program will operate as desired.

It is difficult to solve this problem as long as the inference relies only on the given examples. Therefore, many PBE systems have an interaction session, in addition to a demonstration session, to get more information necessary for the inference. For example, Peridot (Myers 1993) asks the user whether it is applying appropriate rules for the correct inference. MetaMouse (Maulsby and Witten 1993) has the user teach important relationships between graphical objects. Although such interactions are useful for making the proper generalizations, they impose additional burdens on the user. If the cost of program definition is more expensive than that of simply repeating a sequence of operations, users would not use PBE facilities.

4.2.2 Problem of Accessing Internal Data of Applications

Users prefer adding the PBE component on existing applications that they are usually using, rather than replacing their familiar applications with unfamiliar ones in which the basic functions of the application plus PBE functions are embedded.

However, adding the PBE component is not easy. To make the add-on framework available, the external PBE component has to record user operations on the target application and display the program execution results in the application's window. However, ordinary commercial applications do not provide application program interfaces (APIs) that allow external processes to access their internal data or user interface events.

Therefore, PBE systems have to be developed from scratch, including the user interface and the basic functions of applications. Since it is difficult to develop a sophisticated user interface and functions, PBE systems that have been developed in laboratories have not replaced commercial products.

4.3 Web Browsing: Good Domain for PBE

It is difficult to find general solutions to the problems explained in the previous section, but it can be done in specific application domains. One such domain is Web browsing, which is a relatively new domain. Since, prior to the appearance of the Web, no browsing domains that have repetitive tasks were available, many PBE systems only support editing tasks, such as text and graphical editing.

A good feature of the Web browsing domain is that users do not suffer so much from the system's wrong guesses. In the case of editing tasks, wrong guesses may result in important data being missed and/or destroyed, whereas PBE systems for browsing tasks never destroy a user's data. Therefore, the inference engine does not need to be as sophisticated as the engine for the editing tasks.

The data access problem can also be avoided. By tapping into the http stream like a proxy, the PBE component can know the URLs that the user requested and also what the user inputted into HTML forms. It can also add the results of program executions into the original Web pages. This framework can be used in SmallBrowse. (However, in the current implementation of SmallBrowse, PBE functionality is embedded into a browser whose source code is available.)

Another way of capturing user operations is to use the browser's APIs. Fortunately, the widely used Web browsers, Microsoft Internet Explorer and Netscape Navigator, offer various APIs for the external processes. By using these APIs, the external PBE components can identify even the portion of the Web page where the user highlights on the browser window by using the mouse (see the appendix for details). Scrapbook operates by relying on the browser's APIs.

By taking advantage of these features of Web browsing domain, Scrapbook and SmallBrowse are able to be practical PBE systems.

4.4 Internet Scrapbook

World Wide Web (WWW) browsers allow users to access Internet information sources easily. While the operation of these browsers is simple, users need to spend much time and care in their daily access to the Web information they desire for the following reasons:

- The information that users need is usually distributed across several different pages. In cases where users need to obtain a weather forecast, computer news, and sports results from different pages, for example, they have to access all the necessary pages by repeatedly specifying URLs or by selecting them from bookmarks.
- Users often need to browse only a portion of a Web page. On a nationwide weather forecast page, for example, users probably want the forecast only for the area where they live. Consequently, they are often required to search the page for the desired information either by looking or by using the string search function provided by the browser.

S
R
L

If users' target pages are frequently updated, it is a heavy burden for the users to keep up with the latest information by repeating these Web-browsing operations.

4.4.1 Overview of Internet Scrapbook

Internet Scrapbook is a PBE system for automating daily Web-browsing tasks. It clips portions of multiple Web pages and merges the clipped data into a single page so that the user can browse all of his or her necessary information without repeatedly specifying URLs and scrolling to the position where the user's target information would appear.

One characteristic of Scrapbook is its ease of use. The user's main task in Scrapbook is to demonstrate his or her desired information by using actual Web pages. Specifically, the user creates a personal Scrapbook page by repeatedly selecting data in a Web browser (Fig. 4.1[a]) and copying it to the Scrapbook page (Fig. 4.1[b]). The selected data can be copied simply by clicking the Scrapbook's *Copy* button. Once the personal Scrapbook page is created, the user has only to click the *Update* button to browse the latest Web information. The system updates the Scrapbook page reflecting the modifications of the source Web pages (Fig. 4.1[c]).

An important design aspect is that Scrapbook is designed so that Web data can be copied directly from the most commonly used Web browsers—Netscape Navigator 3.0, Microsoft Internet Explorer 3.0, and their newer versions for Windows95/NT3.51—rather than forcing users to use a special browser only for Scrapbook. This can be done using the browser's APIs and Windows' APIs. (The details are given in the appendix.)

Figure 4.2 shows the PBE process in Scrapbook. In the demonstration phase, given a single example of a user selection in the Web browser (Fig. 4.2[a]), Scrapbook infers a matching pattern to identify the selected region in the source Web page (Fig. 4.2[b]).

In the execution phase, Scrapbook updates the user's Scrapbook page by using the generated patterns. It downloads the latest Web page (Fig. 4.2[c]), extracts portions that match the patterns from newly downloaded Web pages, and reconstructs the Scrapbook page with the extracted data. However, a portion that *completely* matches the pattern cannot necessarily be found in the modified version of the Web page. When there is no complete match, the system performs partial matching to find possible candidates for extraction (Fig. 4.2[d]) and selects the most plausible one by applying heuristics (Fig. 4.2[e]).

A major challenge here is how Scrapbook can extract specific portions from frequently modified Web pages in the example-based framework. The

FIGURE **4.1**

FPO Figure 4.1

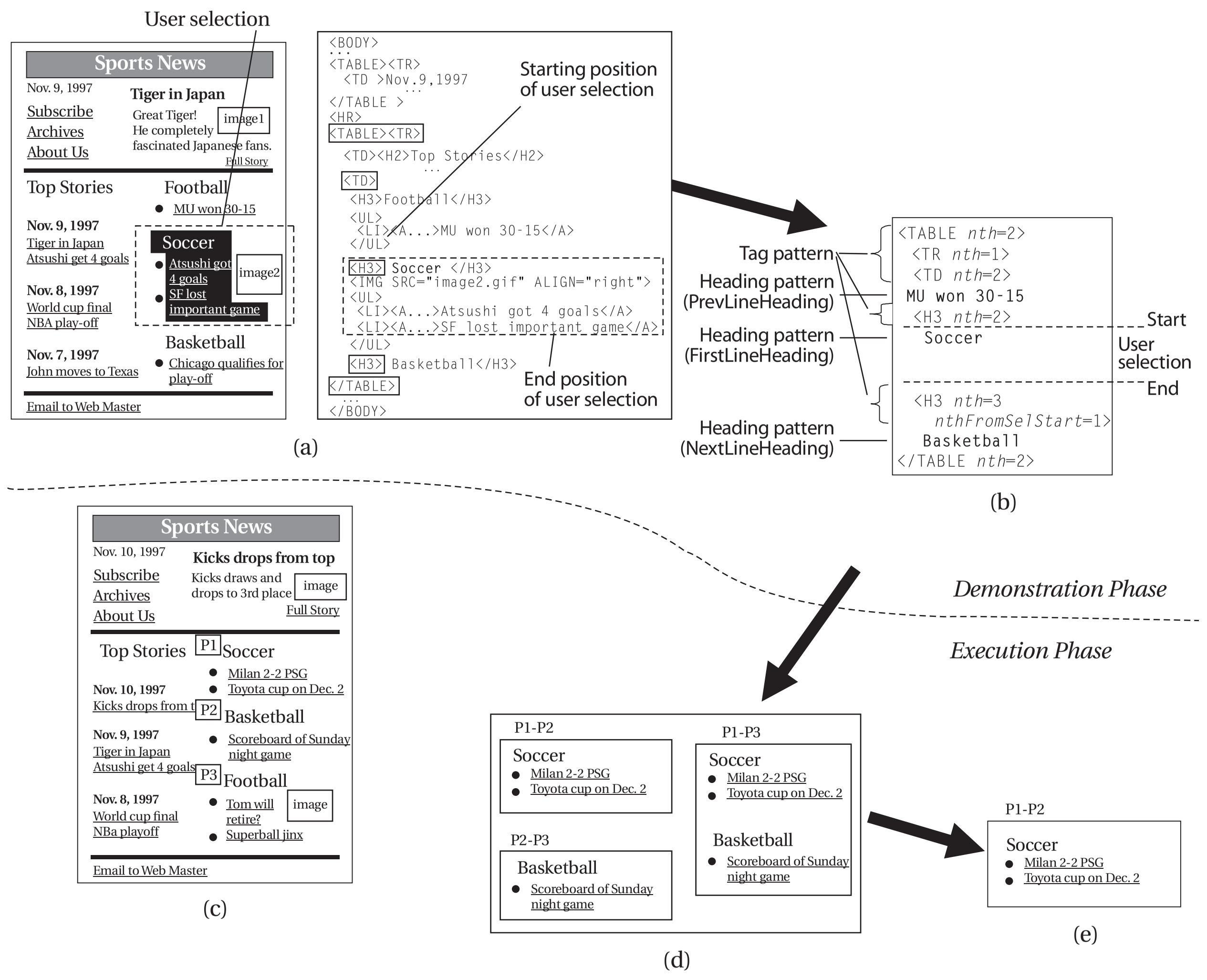
Overview of Scrapbook: (a) user selection in Web browsers, (b) Scrapbook page created by the user; (c) page updated by the system.

system has to find the correct portions on many types of Web pages, using only a single example. To do this, Scrapbook uses a heuristic method based on certain regularities observed in frequently modified Web pages (i.e., that article headings and positions in the Web pages are rarely changed).

4.4.2 Generating Matching Patterns

As mentioned, Scrapbook generates a matching pattern at demonstration time and uses it to extract the user-desired portion from the *future* versions of the page. Therefore, patterns must be described using information that is expected to remain constant even after the page has been modified.

FIGURE 4.2



```
<BODY>
...
<TABLE><TR>
  <TD >Nov.9,1997
      ...
</TABLE >
<HR>
<TABLE><TR>
  <TD><H2>Top Stories</H2>
        ...
  <TD>
   <H3>Football</H3>
   <UL>
    <LI><A...>MU won 30-15</A>
   </UL>
  <H3> Soccer </H3>
  <IMG SRC="image2.gif" ALIGN="right">
  <UL>
   <LI><A...>Atsushi got 4 goals</A>
   <LI><A...>SF lost important game</A>
   </UL>
  <H3> Basketball</H3>
</TABLE>
 ...
</BODY>
```

```
<TABLE nth=2>
 <TR nth=1>
 <TD nth=2>
MU won 30-15
 <H3 nth=2>
  Soccer

 <H3 nth=3
   nthFromSelStart=1>
  Basketball
</TABLE nth=2>
```

S
R
L

According to our observations, there are certain regularities in modifications to Web pages, and two kinds of information are likely to persist over these modifications: the heading of an article and the position of an article. In the Web page shown in Figure 4.2(a), for example, the heading of the selected region "soccer" would not be changed. Also, the top news would always be at the top of that page. Therefore, the system includes descriptions of both the article headings and positions in the matching pattern. These descriptions are called the *heading pattern* and the *tag pattern,* respectively.

Heading Pattern

The heading pattern consists of text that the system regards as the heading of the user-selected article (data). Scrapbook simply infers that headings are likely to be in the lines *previous* to, *first* in, and *following* the user selection. For example, if the user selects the data as shown in Figure 4.2(a), the system generates the heading pattern shown by the bold text in Figure 4.2(b). This heading pattern gives an abstract interpretation of a user selecting the region from "Soccer" (the first line of the selection), whose previous line is "MU won 30–15," to just before "Basketball" (the text following the selection).

This inference is based on some characteristic of the regions selected by users. Users usually select the whole article from the beginning to the end. They do not start in the middle of the article. Since many articles in Web pages are preceded by headings that explain the contents or the category of each article, there is a high probability that the first line of the user selection is a heading.[1] The line previous to the selection may become a heading because of slight variations in the selected region—users sometimes select articles without including the heading. In many cases, the heading of the next article appears in the next line after the user-selected article. The text in that line would be useful for determining the end position of data to be extracted.

Tag Pattern

The tag pattern represents the position of selected data in a Web page. It consists of HTML elements, each of which has an *nth* parameter. The *nth* parameter represents the appearance order of the HTML element counted

1. The system might be able to find headings more precisely if it relied on H1-H6 elements. Since, however, headings are not necessarily tagged by H1-H6 elements (instead FONT and B are often used), we currently employ the simple method of relying on the selected region.

from the document top. For example, the *nth* parameters of the tag pattern in Figure 4.2(b) indicate a region from the second H3 to the third H3 embodied in the second cell (the second TD in the first TR) of the second TABLE.

To identify the starting position of the user-specified portion, the tag pattern contains all HTML elements that mark up the starting position. Only layout elements that affect the page layout, such as TABLE, TR, TD, UL, LI, H1-H6, P, and HR, are included in the pattern, while font style elements, such as FONT and B, are not considered. Since the tag pattern has the nested structure of the HTML elements, the *nth* parameters are hierarchically counted from their parent elements.

To identify the ending position, the *nthFromSelStart* parameter is used in addition to the *nth* parameter. It represents the position relative to the starting position of the user selection, such as the first appearance of an H3 element after the starting position. It is set to the number of elements counted from the starting position.

4.4.3 Extracting Data from Web Pages

Scrapbook extracts the user's target portions from the latest Web pages by using a matching pattern described by the headings and positions of a user-specified article. However, there is no guarantee that both the headings and the positions will remain unchanged, and the pattern might not completely match any portion of the page. For example, if the Web page in Figure 4.2(a) is modified to that in Figure 4.2(c), the pattern in Figure 4.2(b) does not completely match any portion of the page because the position of the "Soccer" section has moved up.

To deal with the various modifications of Web pages, the data extraction process consists of two steps: (1) finding candidate portions of the extraction result by *partial* matching and (2) choosing the correct one among a number of possible candidates by applying heuristics.

Generating Candidate Portions by Partial Matching

Scrapbook finds candidates of starting positions and those of ending positions separately. The candidate regions are generated from all combinations of the starting positions and the ending positions.

If the pattern of Figure 4.2(b) is applied to the Web page shown in Figure 4.2(c), for example, P1 and P2 would be the candidates of the starting position. P1 is the position where the text "Soccer" of the heading pattern exists, and P2 is the position where the tag pattern matches; that is, the user-specified data originally appeared in the document of Figure 4.2(a). Likewise, P2 and P3 would be the candidates of the ending position. They are listed by the heading pattern and the tag pattern, respectively. Combining these candidate positions, Scrapbook makes the list of candidate portions to be extracted: P1-P2, P1-P3, and P2-P3 (Fig. 4.2[d]).

Choosing One Candidate Using Heuristics

Scrapbook chooses one portion from the generated candidates using a set of sophisticated heuristics. Basically, the system prefers portions identified by a heading pattern to those found by a tag pattern. This is because the headings are expected to reflect the user intent in the selection and explain the contents of articles more clearly than the article position. Based on this idea, the region P1-P2, found by the headings "Soccer" and "Basketball," is selected in Figure 4.2(e).

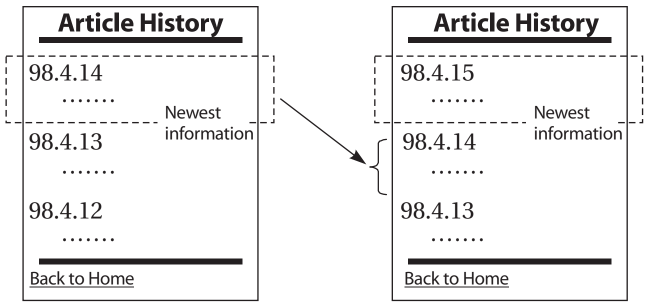
However, things are not quite as simple as this. Imagine the Web page where articles are chronologically ordered with the newest one added to the top, as shown in Figure 4.3. In this case, a portion found by a tag pattern (the article at the top of the page) should be chosen as the extraction result. If the heading pattern is used, the same data from "98.4.14" to "98.4.13" would always be extracted. To handle such a case, Scrapbook has a rule to choose the tag-pattern portion when the heading pattern extracts exactly the same data as before the update. Scrapbook has several other rules to handle various types of Web pages. For more details, see (Sugiura and Koseki 1998).

4.4.4 Evaluation

I asked eighteen subjects to create their own Scrapbook pages by selecting portions of their favorite Web pages, and then update the Scrapbook pages for two weeks. In the experiment, 155 portions were selected and copied to Scrapbook pages by all subjects. They were selected mainly from news sites (any kinds of news, such as general, weather, sports, technical news, etc.), some from company homepages, a music CD sales-ranking page, and novels written in a personal homepage. None of the subjects selected portions starting in the middle of articles.

During the experimental period, a total of 887 updates were made, and 92.1 percent of the updates were successfully made as the subjects desired. According to our observation and the subjects' own views, most of the

FIGURE 4.3



A Web page where the articles are chronologically ordered.

subjects could easily determine whether the update results were appropriate without displaying the whole source of the Web pages for comparison.

This result shows that the headings and positions of articles are very useful for finding the specific portions of Web pages. Although the system could not properly update 8 percent of the selected portions, many of them were trivial failures, such as extracting one line more than expected. We believe that our data extraction method can be of practical use.

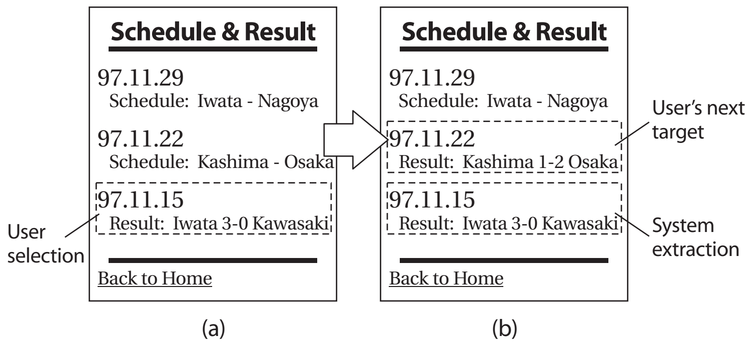
We speculate that there are two reasons that the article headings and positions are preserved. First, changing the structure of Web pages is a heavy burden for information providers as well as costly. Second, Web sites must ensure the readability of their Web pages. If the document structure is changed often, readers cannot easily find their target information.

The robustness of the data extraction method against the various ways of modifying Web pages is supported mainly by the partial matching process for generating candidates of the extraction. The partial matching allows the system to generate a matching pattern without precisely predicting the future modifications of a Web page and to find some candidates for extraction as long as the Web page keeps either the same heading or position of the user-specified article.

Scrapbook fails to extract the user's target portion, if partial matching cannot find any candidate portions in the latest Web page, or if the candidate list generated by the partial matching does not contain the user-desired portion. Finding no candidate portions means that neither the headings nor the positions of the article selected in the user demonstration are preserved. Typically, such cases occur when Web pages are totally rewritten. S

A typical case of no user-desired portion is observed in schedule-result R
pages where the schedules are replaced with the results one by one (Fig. L

FIGURE 4.4



Schedule and result page.

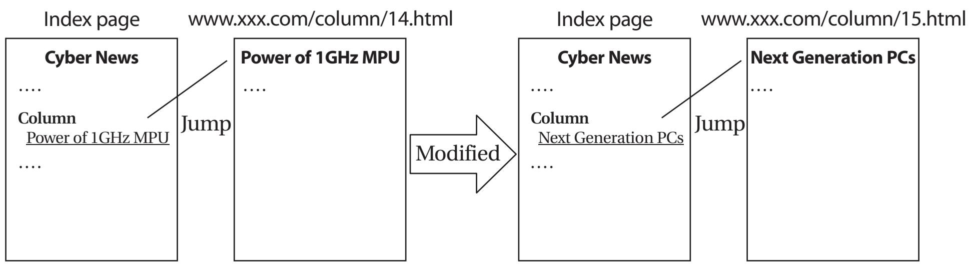
4.4). Since both the heading and the position of the newest result (target information) keep changing, the system can never extract the portion that the user needs.

4.5 SmallBrowse: Web-Browsing Interface for Small-Screen Computers

Recently, palm-top computers and handheld PCs, such as Palm V, have come into wide use. However, document browsing in those computers is not an easy task due to their limited screen size. It is especially difficult to search a document for target data with scrolling, because a user is required to recognize that the data have appeared in the small screen area and immediately stop scrolling before it disappears. In Web browsing, such scroll operations are often required to display a hyperlink to be followed, which is a heavy burden for users. Although bookmarks allow users to jump directly to their target Web pages, they cannot be a solution for the following reasons:

- When many URLs are registered as bookmarks, scrolling operations become necessary again to select one from a long list of bookmarks.
- Bookmarks are effective for recording the pairs of static URLs and page titles, but they are not appropriate in the case where the URL and title of a target page are dynamically changed. Imagine a serial column page

FIGURE 4.5



A URL and a page title dynamically changed.

where each column has a unique serial number, as shown in Figure 4.5. In this case, a bookmark saved at some time point will not be useful next time. News sites often have such dynamic URLs and titles in daily articles. For pages with dynamic URLs, the user actually has to follow a link from an index page to reach the target page. Not only a URL but also an anchor text is important. Since the anchor text is often the same as the title of the referred page, the user needs to see it to determine whether he or she should follow the link.

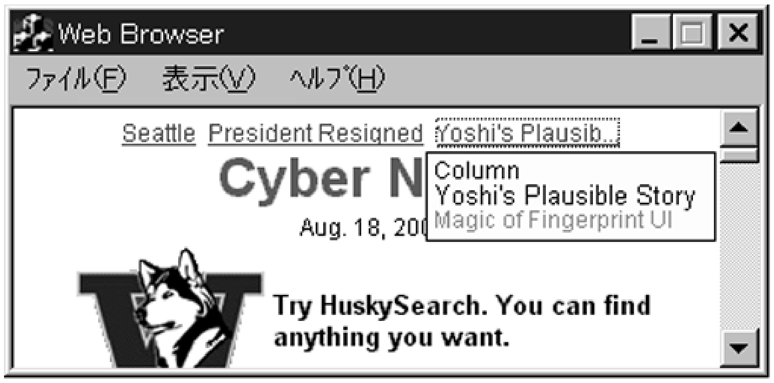
4.5.1 Overview of SmallBrowse

The bookmark problem suggests requirements of a Web-browsing user interface in small-screen computers. That is, only the low number of links that the user is most likely to follow should be shown in the small screen, and at the same time their destination URLs and anchor texts should be refreshed to the newest ones.

Based on this idea, I have developed a PBE component, called *SmallBrowse,* that facilitates Web-browsing tasks in small-screen computers. Using the user's Web-browsing history, SmallBrowse predicts hyperlinks that a user would next follow. Specifically, it chooses three links from the newest version of Web pages and inserts those three predicted links[2] (we refer to them as *p-links*) at both the top and bottom of an original Web page

2. The number of p-links should be determined according to the screen size. SmallBrowse currently assumes a 360×180 resolution.

FIGURE **4.6**


Web Browser
ファイル(F) 表示(V) ヘルプ(H)
Seattle President Resigned Yoshi's Plausib...
Cyber N
Aug. 18, 200
Column
Yoshi's Plausible Story
Magic of Fingerprint UI
Try HuskySearch. You can find anything you want.

Screen snapshot of SmallBrowse.

(Fig. 4.6). Since p-links are extracted from the latest Web pages, their destination URLs and anchor texts are also the newest ones.

The user can use the p-links when the Web page is first displayed in the browser and/or when the user finishes reading the page and jumps to other pages without scrolling. When one of the p-links is focused, SmallBrowse displays a "tip help" to show the data around the original link. This helps the user determine whether he or she should actually follow the link.

Example Task

This section explains an example task on SmallBrowse. The user browses the same news site, which is updated every day, repeatedly on different days. Figures 4.7(a) and (b) shows the first and the second browsing session, respectively.

Since the hyperlink prediction relies on a user's Web-browsing history, p-links are not inserted in the first browsing session. So the user normally follows links. In the index page, the user first scrolls down to the position of the Weather link and follows it. Next in the weather page, she again scrolls to the Seattle link and follows it. After browsing the Seattle weather, she then goes back to the index page using the Back command of the Web browser. Likewise, she follows the link of the column and one of the links of sports articles. The upper portion of Figure 4.8 depicts the sequence of URLs that SmallBrowse recorded through the first session.

In the second session, p-links are inserted based on the recorded history. As shown in Figure 4.7(b), many of the scrolling operations can be eliminated using the p-links. In the index page, for example, Weather, Seattle, and the column links are selected as the p-links because the user followed in the first session these three links just after the index page. Note that the Seattle link can be a p-link for the index page although it is not originally contained in the index page.

Another feature of SmallBrowse's prediction can be seen in the prediction of the column link. The destination URL of column link is continuously changed, and the newest one at the second session (. . . /column/24.html) differs from the one when the user had followed the column link at the first session (. . . /column/23.html). Nevertheless, the column link is selected as a p-link. SmallBrowse judges /column/23.html and /column/24.html as the identical URL by using some heuristics (details in Section 4.5.2.3). This allows the system to handle the continuously changing links.

When multiple links that can be regarded as identical to the old URL are found, SmallBrowse creates a p-link to scroll to the position of the first link found. This can be seen in the p-link inserted into the column page. The p-link Sports leads users to the position of sports articles on the index page.

In the third or subsequent sessions, the Weather link will not be selected as a p-link because it is followed only once in the first session and the user will never visit the weather page. The links followed more frequently, such as the Seattle link and the column link, will be selected as p-links.

Recording a History

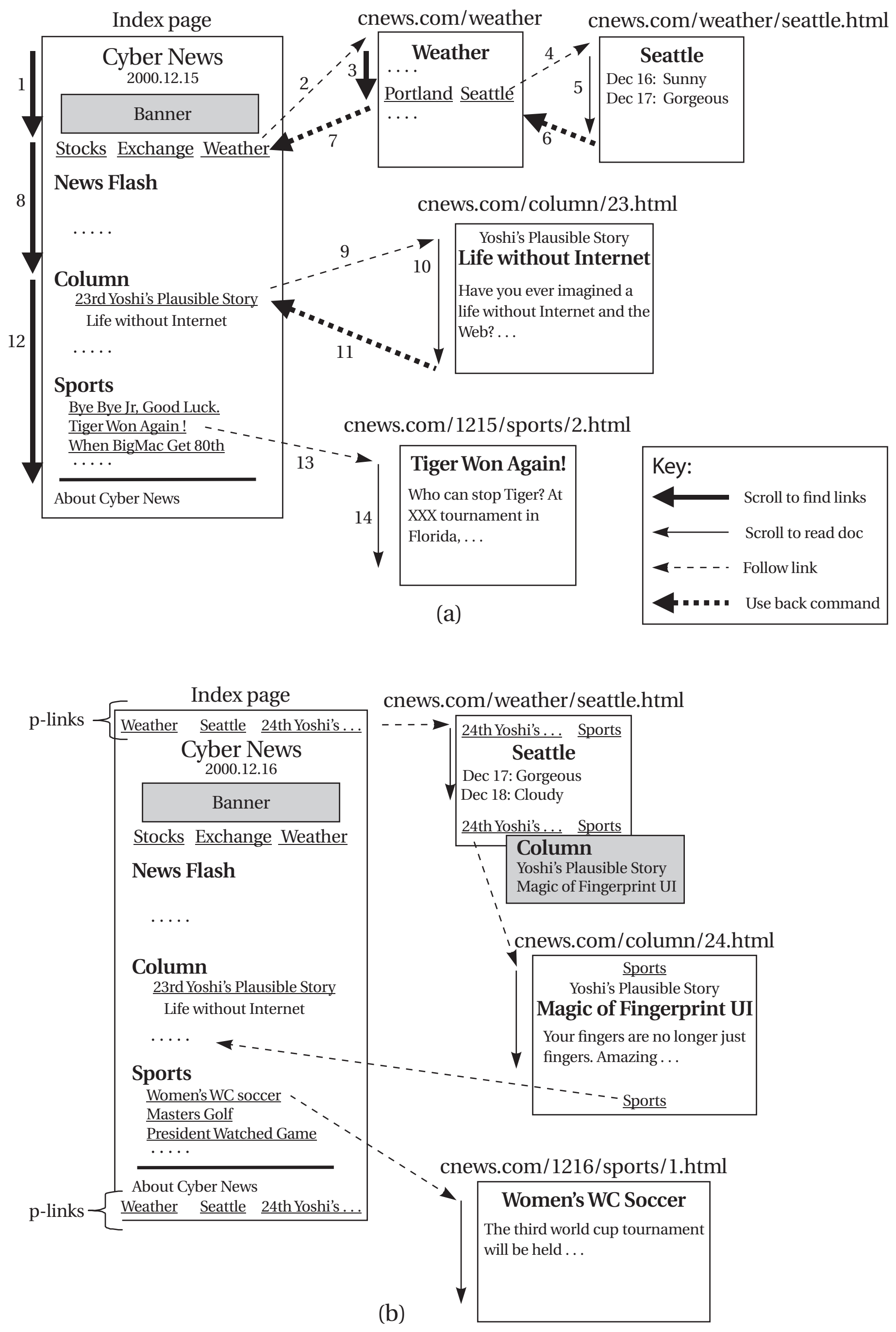
As shown in Figure 4.8, SmallBrowse records pairs of two URLs, called *R-URL* (requested URL) and *S-URL* (link *s*ource URL). R-URL is a URL that the user requested, and S-URL is the latest requested URL of a Web page that contains an R-URL. In other words, the S-URL represents a Web page from which the user jumped to the page of the R-URL. Note Web pages displayed by using the Back command are not recorded in a history.

Predicting p-links

Predicting a p-link means determining a pair of its destination URL and anchor text. So the prediction process consists of two steps: determining the URL of a p-link and finding in the latest version of a Web page an anchor text to be associated with the predicted URL.

In the first step, a URL is predicted only by a sequence of R-URLs in the history. In the second step, the anchor text is extracted from S-URL pages.

FIGURE 4.7



Web browsing on SmallBrowse.

FIGURE **4.8**

Web pages that a user browsed	**Web pages that had been containing the browsed URLs**	
http://www.cnews.com/	N/A (specified from bookmark)	First session
http://www.cnews.com/weather/	http://www.cnews.com/	
http://www.cnews.com/weather/seattle.html	http://www.cnews.com/weather/	
http://www.cnews.com/column/23.html	http://www.cnews.com/	
http://www.cnews.com/1215/sports/2.html	http://www.cnews.com/	
. . . .		
http://www.cnews.com/	N/A (specified from bookmark)	Second session
http://www.cnews.com/weather/seattle.html	http://www.cnews.com/weather	
http://www.cnews.com/column/24.html	http://www.cnews.com/	
http://www.cnews.com/1216/sports/1.html	http://www.cnews.com/	
. . . .		

Web-browsing history.

Since some URLs are continuously changing, however, the second step is also used to refresh the predicted URL (URL selected from the R-URL history) to the newest one. To do this, SmallBrowse performs pattern matching by using an abstracted description of the URL.

Determining URLs from a History

To insert p-links into a Web page of a URL U0, SmallBrowse selects as URLs of those p-links the most frequently requested URLs after U0. Specifically, the system first finds U0 in the R-URL history, then picks up the next three URLs, and counts the occurrences of each URL. The URLs with the top three occurrences are selected for the URLs of p-links. Consider a case, for example, where the R-URL history is as follows:

U0 *U1 U2 U3* U4 U5 U0 *U2 U3 U4* U5 U0 *U1 U3 U5* . . .

In this case, U3 (three occurrences), U1 (two occurrences), and U2 (two occurrences) are selected as the URLs of p-links that are inserted into the Web page of U0.

However, URLs of some Web pages are changing continuously. In the example task, a URL of the column page contains a serial number, and the URL for the newest one is different from any of the recorded R-URLs. Therefore, the column link would never be chosen as a p-link if the system counted the occurrence of URLs by simply comparing their strings.

To solve this problem, SmallBrowse uses simple heuristics about formats of URLs of periodically updated articles. Such URLs are likely to contain decimal numbers that represent year, month, day, and/or a serial number. For example, URLs for the newest columns might be changed as follows:

http://www.cnews.com/column/<u>23</u>.html (the 23rd column)
http://www.cnews.com/column/<u>24</u>.html (the 24th column)

SmallBrowse regards two URLs as identical when the differences between them are only decimal-number parts. In other words, SmallBrowse compares URLs at an abstract level. In the case of column page, both of the above two URLs can be generalized into "www.cnews.com/column/\d*.html," and they are dealt with as the same one. The occurrences are also counted based on the abstract patterns of URLs.

Finding the Newest URL and Anchor Text

The next task is to find anchor texts to be associated with the predicted URLs. To reflect the current state of an anchor text, SmallBrowse looks through the latest version of the C-URL page corresponding to the predicted URL (the one selected from the R-URL history) and extracts a link whose destination URL matches the abstract pattern of the R-URL. Note that through this process the URL as well as the anchor text are refreshed from the old state URL, recorded in R-URL history, to the newest one.

Imagine a case where the system is predicting p-links in the second browsing session based on the R-URL history recorded in the first session (Fig. 4.8), and it selected "www.cnews.com/column/23.html" as a URL of a p-link. In this case, the system searches the current "www.cnews.com/" page (S-URL corresponding to the selected R-URL ". . . /column/23.html") for links that match the generalized pattern of ". . . /column/\d*.html." As a result, ". . . /column/24.html" matches the pattern. That newest URL and its anchor text will be used in a p-link.

One special treatment takes place when there are multiple URLs that match the abstracted pattern. In this case, the system generates a URL to scroll to the position of the first matched URL in the C-URL page (using <A> NAME property) and assigns as its anchor text a heading text just before the first matched URL. The heading is identified using some heuristics about font size, font color, horizontal rule, and so forth.

Such multiple matching cases often take place when the articles of the same category are listed. In the example task, this can be seen in sports articles of the index page. The system inserts a p-link to directly jump to the position of a list of sports articles and sets its anchor text to "Sports," which is the heading of those articles. Such p-links gives the user a chance to select from the list of the articles the ones that the user wants to read.

4.5.2 Tip Help

In some cases, an anchor text does not explain the content of the reference. In the column link in the example task, the format of its anchor text is “n-th Yoshi’s Plausible Story.” The content of the referred document is explained in the next line of that link (Fig. 4.7).

SmallBrowse shows data that seem related to the content of the reference with a tip help. The system identifies headings just before and after the link and extracts text between two identified headings. This feature would be helpful for users.

4.5.3 Informal Experiments

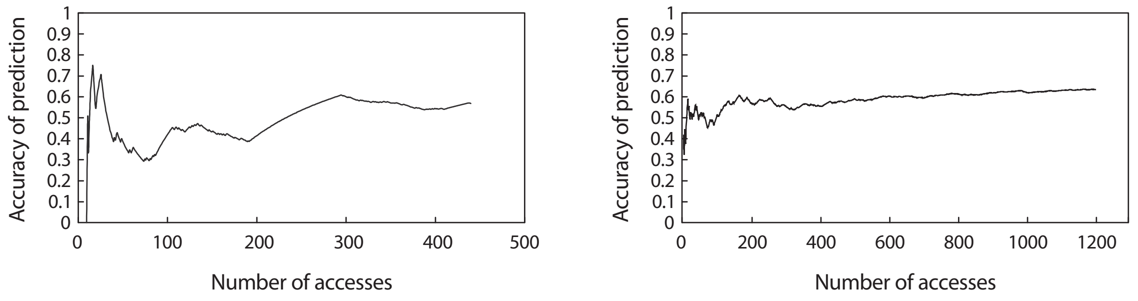
The author conducted informal experiments to confirm the effectiveness of the prediction method of SmallBrowse. Unfortunately, the current version of SmallBrowse is not sufficiently sophisticated to be used for user study. Therefore, I simulated the prediction process using access logs recorded by a proxy and estimated the accuracy of p-links’ URLs. That is, I predicted three URLs (destination URLs of p-links) that the user would next access for each URL in the access log and examined whether one of the three predicted URLs was the one that the user actually requested next.

In the experiments, access logs of six subjects over twenty-eight days were used, and the accuracy was calculated for each subject. During the experiments, 7,350 accesses were made in total (one subject accessed about forty-four pages per day on average), and 50.2% of URLs were requested more than once. The predictions were done on these repeatedly requested URLs. Using the access histories for twenty-eight days, the accuracy of predictions for each user was 31.7%, 40.5%, 42.9%, 56.8%, 63.3%, and 64.2%. Figure 4.9 shows the typical learning curves on two subjects.

4.6 Discussion

Although the application domain of Scrapbook and SmallBrowse is the same, the strategies they use to achieve a practical level are different. The

FIGURE **4.9**



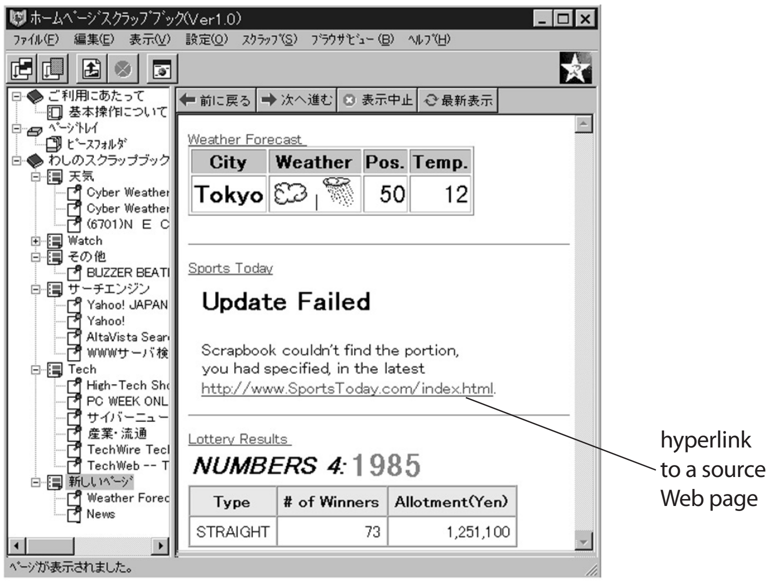
Accuracy of predictions: (a) subject a and (b) subject b.

Scrapbook module has its own user interface (it can also be used in combination with Netscape Navigator or Internet Explorer) and requires users to give examples explicitly through the demonstration session. This means that some cost for learning how to use the system and for instructing the system is imposed on users, although the operations in Scrapbook are simple. To compensate for this, Scrapbook pursues high accuracy in extracted user-specified portions from the newest Web pages (about 92 percent).

In SmallBrowse, the accuracy in predicting hyperlinks is at most 64 percent. However, it does not require users to do anything special to use the prediction results. Namely, since the prediction is based on the user's Web-browsing history, the user does not have to invoke special commands for recording the user's browsing actions. Also, the prediction results are inserted into Web pages as p-links that can be used in the same way as the ordinary links. That is, the users do not have to learn anything about how to use SmallBrowse. They have simply to click the p-links to jump to other pages just as when clicking ordinary links. Because of this advantage of SmallBrowse, we could expect that users would be generous when it comes to the system's failures.

No matter which strategy is taken, users do not seriously suffer from the systems' (Scrapbook's and SmallBrowse's) wrong guesses. As mentioned in the Scrapbook section, even in the case in which Scrapbook cannot exactly extract portions of Web pages as the user desired, many of the failures are trivial, such as extracting one line more than expected. Even when the system cannot find any portion to be extracted, it displays a message as shown in Figure 4.10 so that users can immediately access the source Web page by clicking the hyperlink. In SmallBrowse, users simply have to perform

FIGURE 4.10


ホームページスクラップブック(Ver1.0)
ファイル(F) 編集(E) 表示(V) 設定(O) スクラップ(S) ブラウザビュー(B) ヘルプ(H)
前に戻る 次へ進む 表示中止 最新表示
hyperlink
to a source
Web page

Weather Forecast

City	Weather	Pos.	Temp.
Tokyo		50	12

Sports Today

Update Failed

Scrapbook couldn't find the portion,
you had specified, in the latest
http://www.SportsToday.com/index.html.

Lottery Results

NUMBERS 4: 1985

Type	# of Winners	Allotment(Yen)
STRAIGHT	73	1,251,100

ページが表示されました。

Message for failure of data extraction.

browsing operations as usual even when the system fails to predict appropriate links.

I believe that both of Scrapbook and SmallBrowse are or will be operational on a practical level. Even after the experiments on Scrapbook had finished, some of the subjects continued using Scrapbook. In fact, Scrapbook is being preinstalled in NEC PCs and is in use by many endusers. As for SmallBrowse, user studies have not been done yet, but I think that the framework of SmallBrowse, which does not impose any burdens on users, is acceptable for many users.

Since the invention of Web, many researchers studying PBE, autonomous software agents, artificial intelligence, and so forth, have focused their application domain on the Web.

The research area to which Scrapbook is the most relevant is wrapper induction for extracting a specific portion from a Web page. Trainable information assistants (TrIAs; Bauer and Dengler 1999) are software agents that use a PBE technique to create wrappers. They are aimed at very accurately extracting a small piece of information, such as the price of a specific airline ticket. To make this possible, TrIAs employs an interactive session, in addition to a demonstration session, to obtain information necessary for generalization through various types of dialogues with a user.

Scrapbook, on the other hand, is designed to extract a relatively large block of Web data such as one whole line or article or a whole list of hyperlinks. Another feature distinguishing from TrIAs is that Scrapbook does not have an interactive session because users do not prefer additional tasks. Actually, Scrapbook used to have an interactive session (Sugiura and Koseki 1998), but only a few users were willing to use the interactive learning framework. So we decided not to include it in our commercial version of Scrapbook. It is, of course, critical to be able to extract the user-specified portions exactly. However, more important things are ease of use and simplicity of operations.

The concept and architecture of SmallBrowse are similar to that of Letizia (Lieberman 1995) and WBI (Barret, Maglio, and Kellem, 1997), which are autonomous interface agents that assist Web browsing based on the user's past Web-browsing history. Letizia records the URLs chosen by the user and reads the pages to learn the user's interests. The interest is represented by pairs of terms and their frequencies. When the user is browsing the Web, Letizia recommends a Web page that is the most likely to match the user's interests by searching "nearby" the user's current position (a set of pages linked from the current page). Since Letizia needs a large screen area to display its recommendation, however, it is not suitable for browsing on small screens. WBI inserts shortcut links by analyzing the user's usual browsing pattern. Unlike SmallBrowse, however, WBI does not have a function to refresh URLs and anchor texts for the shortcut links.

4.7 Conclusion

This chapter described two PBE systems for Web browsing. Web browsing is a good application domain of PBE because of its highly repetitive nature. PBE can clearly offer an advantage to a wide range of users of Web browsers. Furthermore, the failures of inferences cause only minor inconvenience, compared with other tasks, such as text editing and user interface construction.

PBE has been applied to desktop applications so far. However, I believe that a wide variety of application domains are good for PBE. One example is the small-screen, portable computer domain, which SmallBrowse is targeting. Another possibility is the wearable computer domain. Such environments have a natural need for intelligent support from a PBE component.

Appendix: Copying HTML Data from Web Browser to Scrapbook

In Scrapbook, Web data can be copied directly from Netscape Navigator and Microsoft Internet Explorer to a user's Scrapbook page, as shown in Figure 4.1(a) and (b). When HTML data are copied, not only the text selected in the Web browser but also the HTML elements that affect that text need to be copied. However, the system can only get the selected text through a cut buffer (Windows clipboard). Scrapbook copies the related HTML elements in the following fashion:

1. Get the text, which the user selected in the browser, via the cut buffer.
2. Get the complete source of the HTML document through the API provided by the browser.
3. Find the starting and ending position of the user selection in the source document obtained in step 2 by searching the source document for the selected text of step 1. In this search, HTML elements in the source document are not considered.
4. Parse the HTML of the source document, and pick up the HTML elements that affect the region identified in step 3.
5. Copy the text and the HTML elements within the region found in step 3, together with the related HTML elements picked up in step 4.

In step 5, all HTML elements within the region selected by the user are copied into Scrapbook. This means that objects, such as inline images, Java applets, plug-ins, and ActiveX controls, can be copied if the user selects text surrounding those objects, although the objects themselves are not selected in the browsers.

References

Barret, R., P. P., Maglio, and D. C. Kellem. 1997. How to personalize the Web. In *Proceedings of CHI'97.*

Bauer, M., and D. Dengler. 1999. TrIAs: Trainable information assistants for cooperative problem solving. In *Proceedings of the 1999 International Conference on Autonomous Agents* (Agents).

Cypher, A., ed. 1993. *Watch what I do: Programming by demonstration.* Cambridge, Mass.: MIT Press.

Maulsby, D., and Witten, I. H. 1993. MetaMouse: An instructible agent for programming by demonstration. In *Watch what I do: Programming by demonstration,* ed. A. Cypher. Cambridge, Mass.: MIT Press.

Myers, B. A. 1993. Peridot: Creating user interfaces by demonstration. In *Watch what I do: Programming by demonstration,* ed. A. Cypher. Cambridge, Mass.: MIT Press.

Lieberman, H. 1995. Letizia: An agent that assists Web browsing. Paper presented at the International Joint Conference on Artificial Intelligence, Montreal, August.

Sugiura, A., and Koseki Y. 1998. Internet Scrapbook: Automating Web browsing tasks by demonstration. In *Proceedings of UIST'98.*