The following paper was originally published in the
Proceedings of the USENIX Symposium on Internet Technologies and Systems
Monterey, California, December 1997

Creating a Personal Web Notebook

Udi Manber
University of Arizona

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org/

Creating a Personal Web Notebook

Udi Manber[1]

Department of Computer Science
University of Arizona
Tucson, AZ 85721
udi@cs.arizona.edu
http://glimpse.cs.arizona.edu/udi.html

ABSTRACT

This paper introduces a tool, called Nabbit, to go from the World-Wide Web to Your Own Web. Nabbit uses a copy-and-paste paradigm, adapted to the way the web is used, to provide a convenient personal notebook. While browsing, users can select, with the mouse, any part of an HTML page they are looking at, and Nabbit will copy that part with the original format — images, forms, links and all — to their own pages. The source, date, and even personal comments are copied as well. Collection of information becomes as simple as ''here's what I want — click — I got it.'' Nabbit can be used to write reports interleaved with web content, maintain extended hot lists (with your own comments and even parts of pages), collect selected hits from search results, and much more.

1. Introduction

How do you remember something you have seen on the web? Besides writing the information down by hand, there are currently only two major methods for keeping track of web information: Adding the current page to a hot list (or bookmarks, or favorite list), or using the SaveAs command to save the contents of the page. Both methods work well for small number of pages, but they do not scale. Anyone who uses the web extensively is running against this problem.

Several methods have been suggested and employed. Our Warmlist tool [1] extends the hot-list concept by automatically saving the full text of all hot-list entries and providing search. There are tools that capture *everything* you load and allow you to search and browse the full history of your browsing. The original Mosaic had annotation features [2], which somehow did not catch on as much as they should have.

The main issue here is convenience. The web itself is so popular because of convenience — with a few clicks one can get the world. However, the process of collecting relevant information and putting it together cannot be fully automated. Saving files cannot be completely transparent, because what to save depends on the users. If you save too much you run into the same scale problems when you need to use that information. If you save too little, or if it is too cumbersome to save, you lose information.

We need tools that give users the power to decide what exactly to save and what to do with it, but let them do it so conveniently that it does not interfere with their normal browsing. This is exactly what Nabbit is designed to do.

---

[1] Supported in part by NSF grant CCR-9301129, and by the Advanced Research Projects Agency under contract number DABT63-93-C-0052.

The information contained in this paper does not necessarily reflect the position or the policy of the U.S. Government or other sponsors of this research. No official endorsement should be inferred.

Nabbit works with two Netscape windows, one for browsing and one for collecting information; in other words, one for input and one for output. (Nabbit is implemented at the moment only for Netscape running on UNIX.) The output window can be iconized and out of view. Let's say that you want to collect information about a certain topic and you use a search engine. Not all the hits it gives you will be relevant. You would like to select some of them and remember only those. With Nabbit, all you do is select with the mouse the part of the page you want to remember, and click on the ''Copy'' button on Nabbit's window. You may select hits number 4, 6, and 7, then follow one of the links, and copy half of that page, go back to the search results, get another set of hits and select hits 14, and 18, follow several links to maybe a local search engine, copy that form (so you can perform searches on it later on), copy a table, a set of links, email addresses, and an interesting paragraph. Everything you copy is being assembled into one HTML page, which you can view on the output window. All copying is done simply by selecting the part you want with the mouse and clicking on Copy. Links to the original sources, dates, and optional mirror copies are automatically added. You can save (publish) that page at any time, you have unlimited ''undo''s, you can (full text) search all the pages you collected, you can load an old page to the output (or input) windows and add to it, and so on.

Everything that Nabbit does can, of course, be done with other means. For example, a good HTML editor (such as the one that comes with Netscape 3 Gold, or IE 4.0) allows copying of parts of pages. But it is inherently more complex for the user, because it is not integrated with the browsing. The page needs to be saved into a file, the editor needs to be started, the required part needs to be copied to another place, and only then can the browsing proceed. The user is distracted enough not to do it on a regular basis. With Nabbit this whole process takes one click.

Examples of the use of Nabbit are given in the Appendix (and the reader may want to jump there early). We first describe the main algorithm behind the capabilities of Nabbit.

2. The Main Algorithm

The heart of Nabbit is a copy-and-paste paradigm. To move HTML code from one place to another, Nabbit requires only that you select what you see on the browser's window. Nabbit then takes the selection (which is always simple text) from the clipboard and figures out the appropriate HTML code for it. In a nutshell, it works as follows: In addition to the clipboard, Nabbit also fetches the HTML source of the current page (using Netscape's remote command facilities, described later). Given a text selection and a source HTML, Nabbit extracts the text from the HTML code, and then employs an *approximate string matching algorithm* to find where the selection best matches the text. Once the location of the selection is found in the HTML code, only the tags that are relevant to that selection are taken, forming a stand-alone HTML piece that corresponds to the original selection as it looked on the browser. This part is not easy, because (practical) HTML is not as clean as it looks. (Actually, HTML often doesn't even *look* clean.) The new HTML piece is then shown on the output window, added to whatever is currently present in the output window, or is used by Nabbit as a base for fetching more documents. That's the essence of the algorithm. Let's go into a few more details.

Let's call the text selection that is copied to the clipboard T, and the source of the HTML document S. Both T and S are strings of characters. We need to find the location in S that generated the text T. In general, all the characters in T appear somewhere in S, although there are a few exceptions. One obvious exception is white space, which may be generated by some HTML tags (like <P> or <BR>). Another, less obvious, exception is list numbering generated by the <OL> tag. These

numbers will be copied to the clipboard, but of course they are not explicitly in S. (The bullets generated by <UL> are not copied to the clipboard, by the way.) There are also many examples of characters in S that are not in T. They include formatting commands, HTML tags, special characters, white space, and more.

To find the source of T in S, we first parse S to divide it into HTML tags and text. The general rule is that everything between the <> brackets are HTML tags and everything else is text. Again, there are exceptions. For example, the content of OPTION tags are outside the brackets but they do not appear as text in the browser and they cannot be copied. The TITLE tag is another example. We strip all white space, because the correlation it contributes is generally low. We then compare T to the text in S, but do so *approximately*. That is, we allow insertions and deletions both in the text and in T. The algorithm we chose is not a well-known one [3]; it allows to set arbitrary costs for each insertion and each deletion based not only on the characters but also on their location in both strings. (To be honest, another reason for this choice was that the code was immediately available to us.)

After the approximate string matching is performed and a location of T is found in S, we need to reconstruct the HTML formatting. Our first attempt was to perform a complete parsing of HTML and then to re-build the selection HTML from that. While this looks like the right approach theoretically, in practice it did not work well. We found that a large percentage of HTML pages on the web — even pages generated by ‘‘authoring programs’’ — contain major HTML errors. We could not afford to simply output ‘‘HTML error’’ (like compilers do) and quit. Instead of trying to fix those errors, we decided that the best solution is to leave them in! After all, when you copy something you would like it to appear in the same way. If the errors are left untouched, they will be handled on the copy in the same way they are handled in the source. This turned out to work very well.

Overall, the algorithm consists of 6 steps:

1. remove white space from both T and S,
2. divide the HTML into tags and text, and store the original positions of both,
3. find the location of T in the text part of S (allowing up to 50% insertions or deletions),
4. determine which tags have no effect on T and can be ignored,
5. put the relevant tags back in their original place around T,
6. add extra information (such as a link to the source, and date).

Since web pages are almost always pretty short, and the network is (still) almost always relatively slow, all these processing steps are negligible in terms of running times. (The pattern matching part of the algorithm, which is the most CPU intensive, is written in C for best performance.)

Although we do not try to understand the structure of a page, we do make some attempts to ensure that the resulting HTML code is good. Here are some important examples.

- We close all open tags. If a certain page contains an <A> tag (link) or a <B> tag (bold) that were not closed (not uncommon), we don't want them to affect everything after the copy.
- If the selection contains any part of a FORM, we copy the whole FORM. Partial FORMs are not always workable, and it's not always clear from looking at the browser where the FORM begins and where it ends.
- We do allow copying parts of tables. In that case, we try to make the partial table as close in formatting to the original table as we can. Unlike FORMs, partial tables, although they may look awkward, can be very useful.

3. The Interaction with the Browser

Nabbit communicates with Netscape through Netscape's remote control mechanism [4], a wonderful yet not widely known facility provided by Netscape only in its UNIX versions. (We are currently working on ports to other platforms, notably Windows and IE, which will require different communication mechanisms. We believe that the same approach will work, with the communication done through the browsers' APIs, although the code will be more complex.) The remote control mechanism allows activation of Netscape menu items on any Netscape window from another process. For example, one can save the source HTML of the current page to fileName by issuing

```
netscape -id id_number -remote
saveAs(fileName)
```

where id_number is the window id given to the netscape window by the X window manager. Similarly,

```
netscape -id id_number -remote
openFile(fileName)
```

brings the contents of fileName into the netscape window. These are the only two actions we need. We use the saveAs mechanism in lieu of fetching the content based on a URL for two reasons. First, it is faster, because it usually copies the content directly from memory and there is no need to go again to the network. Second and just as important, it allows Nabbit to work on results of searches that used the POST action. Such results cannot be fetched from the URL. The communication between Nabbit and Netscape was mostly borrowed from NetShell [5].

When Nabbit is started, it obtains from the X window manager the list of all netscape windows (using the xwininfo program) and present them to the user to choose which one will be the input and output windows. The appropriate id_numbers are then used. This choice can be changed at any time. The X window manager is also used by Nabbit to obtain the URL of the current page. We found no easy way to get that information from Netscape (it is not available in any menu item). But Netscape does tell the X window manager the URL so it can be shown in its box. The URL is not essential to Nabbit's operations, but it is useful to include it as part of the copy, so users can go back later to the source.

Overall, even though the interprocess communication between Netscape, the X window manager, and Nabbit is currently mostly ad-hoc, it is extremely effective and easy to use.

4. The User Interface

The command window of Nabbit is shown in Figure 1 (it looks much better in color). The ‘‘copy now’’ button does most of the work. The ‘‘File list’’ area that occupies the middle part of the window shows the files that were saved before, and it allows to load any of them (or any other file) to either the output or input windows. The ‘‘Where’’ and ‘‘What’’ selection menus are shown in Figure 2. They give several options of what to copy and where to put it. The Notes button opens a text window into which the user can type (or copy) any notes they wish to add.

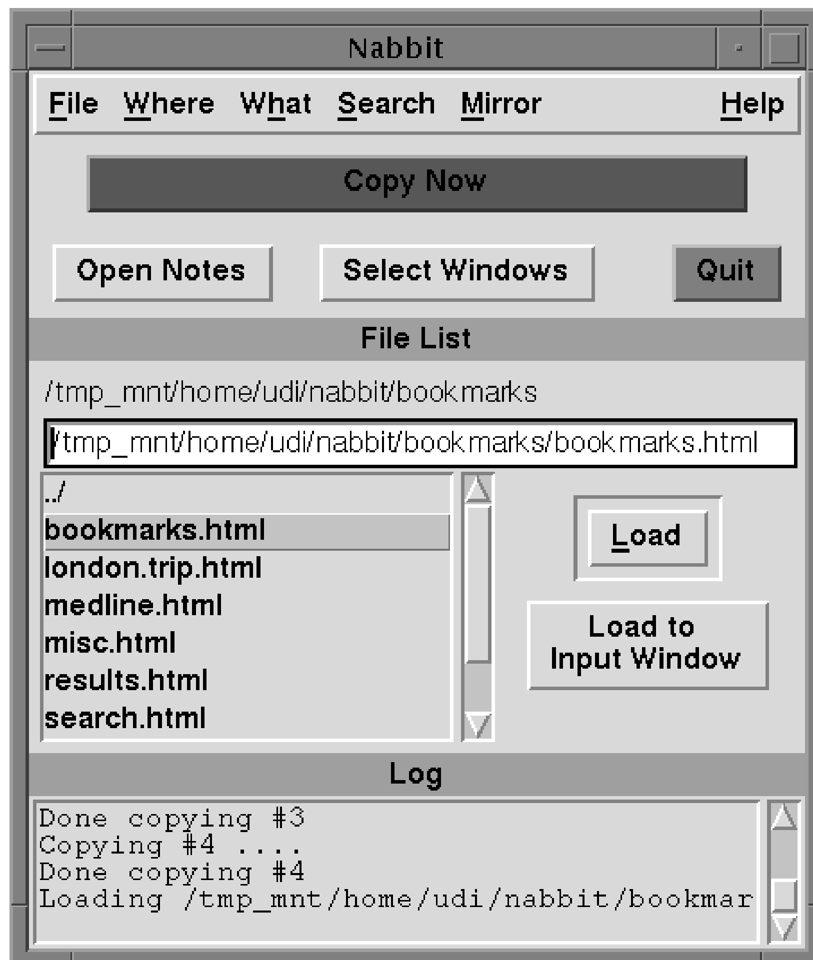

Figure 1: Nabbit's user interface

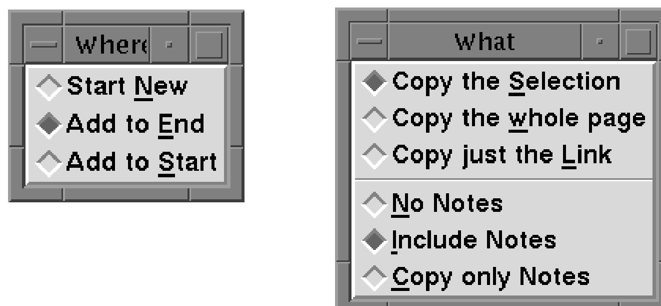

Figure 2: The Where and What options

Other menu items include Search — using glimpse — and Mirror, which mirrors into the local disk the current page and/or pages and images pointed from that page (only one page mirror is currently implemented).

Since every copy involves loading a different page to the browser, we get a very nice side effect of having unlimited undo's! To undo a copy, simply press ''back'' on the output window.

5. Conclusions and Further Work

Nabbit provides a convenient and natural way to take notes while browsing the web. As the web is becoming the primary interface to information, it is essential to find better ways to capture that information. Besides just copying parts of regular web pages, Nabbit can be effectively used to collect results of database searches, which will be more and more important as more databases are connected to the web. Our next development step is to extend the publishing capabilities of Nabbit, allowing people to easily publish their ''notes'' in their organizations, intranets, or throughout the web. This will provide another way to collaborate and use the web more effectively.

6. Acknowledgements

David Lipman and Jim Ostell of the National Center of Biotechnology Information at NIH triggered this work by suggesting to me the problem and its applications, and contributed many useful comments.

References

[1] P. Klark, and U. Manber, ''Developing a Personal Internet Assistant,'' *Proceedings of ED-Media 95, World Conf. on Multimedia and Hypermedia*, Graz, Austria (June 1995), pp. 372–377.

[2] Mosaic User's Guide: Annotations, http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/help-on-annotate-win.html

[3] U. Manber and S. Wu, ''Approximate String Matching With Arbitrary Costs for Text and Hypertext,'' *Proc. of the IAPR International Workshop on Structural and Syntactic Pattern Recognition*, Bern, Switzerland (August 1992), pp. 22–33.

[4] Zawinski, J., Remote Control of UNIX Netscape, http://home.netscape.com/newsref/std/x-remote.html (December 1994).

[5] D. Zhang, and Udi Manber, ''NetShell — Customized Handling of WEB Information,'' http://www.cs.arizona.edu/netshell (June 1996).

Appendix: Examples

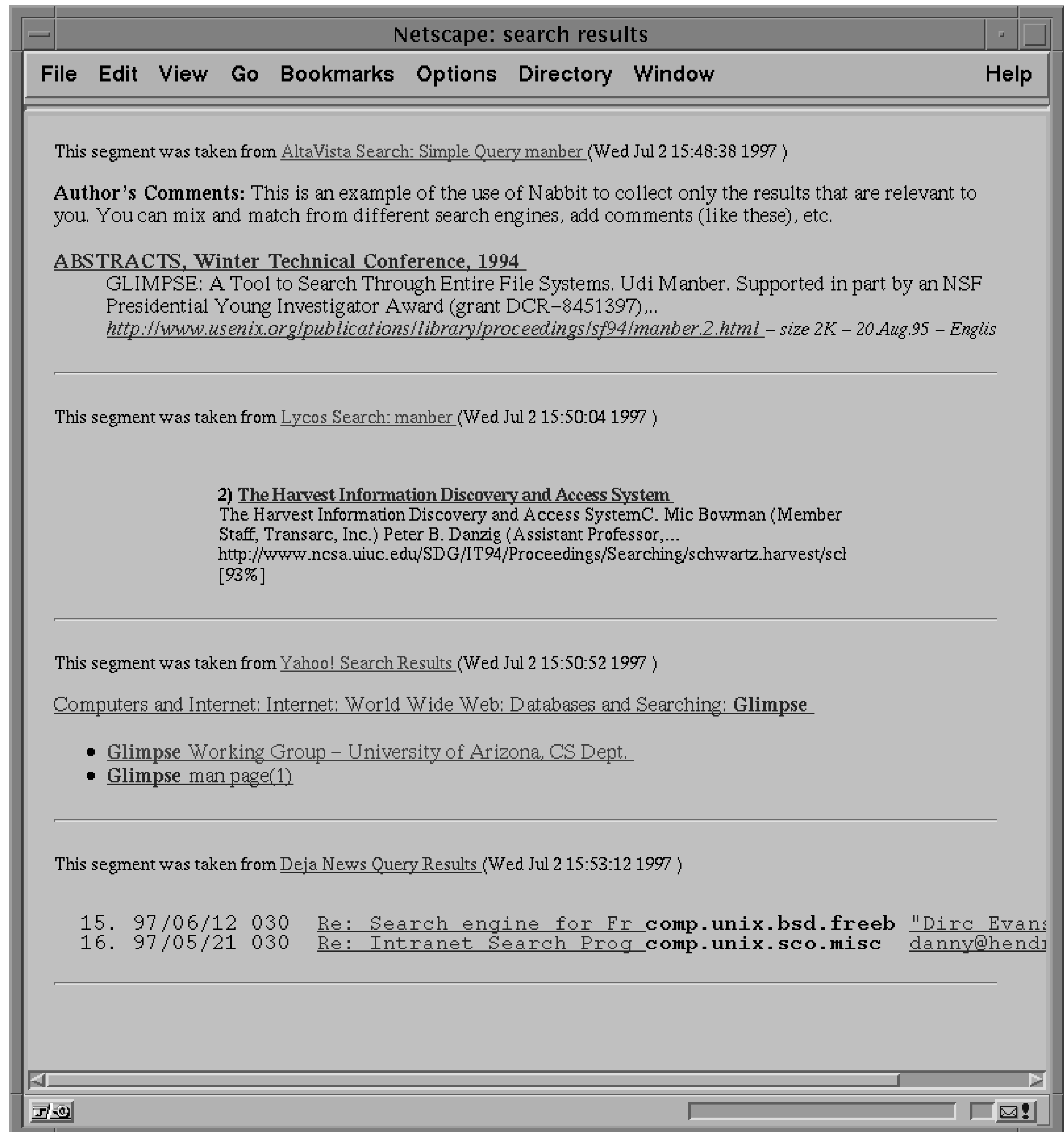

Figure 3: Examples of collecting Search Results

Figure 4: Examples of collecting Search Forms

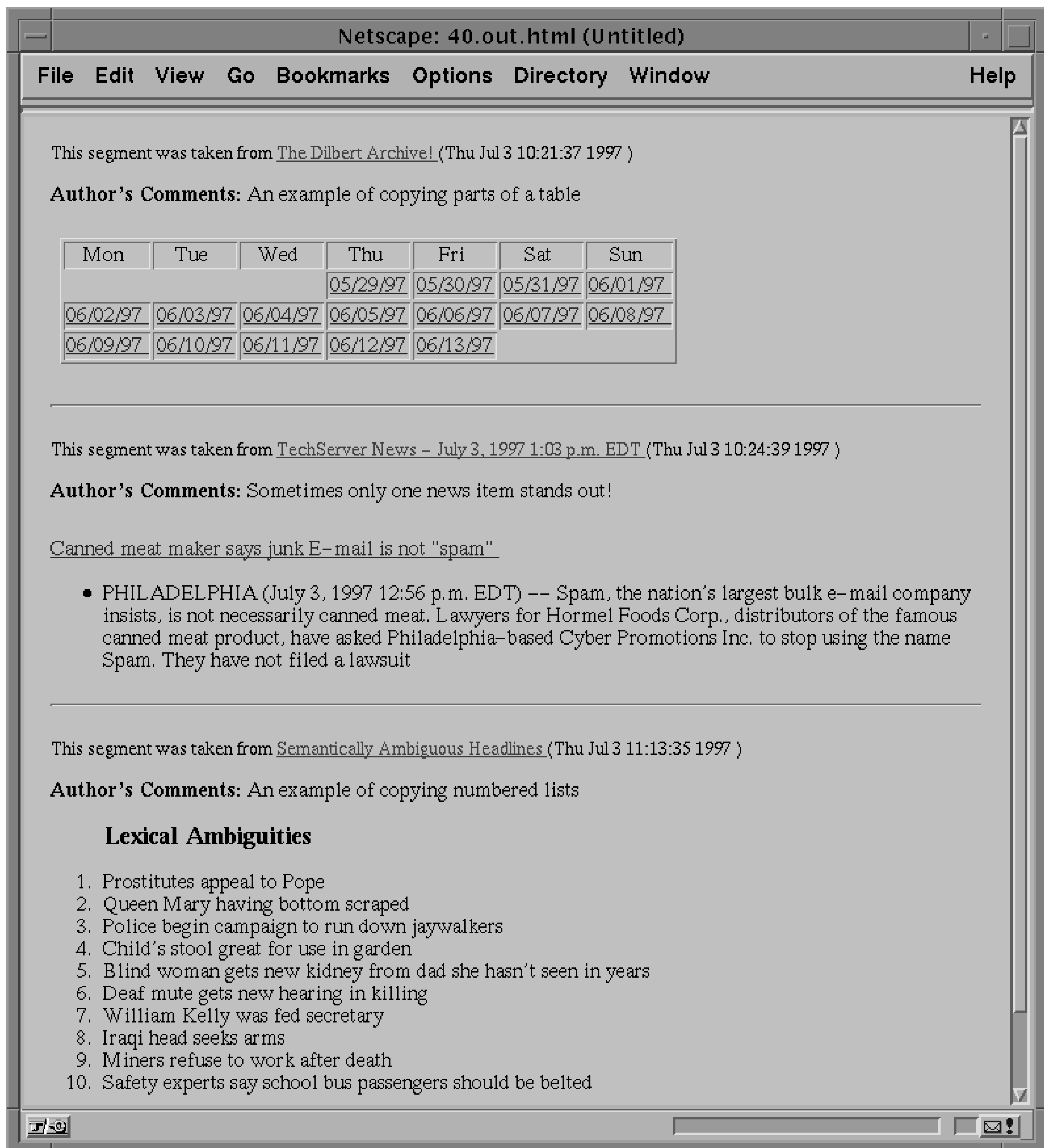

Figure 5: Miscellaneous Examples