Reading newspapers on the web with a refined history feature

Theodorich Kopetzky
Linz, Austria
Email: theoky.ctxt@gmail.com

Abstract

Current web browser offer a history feature. Interestingly, this feature can still be refined. In this paper such a refinement is presented: the *history of the seen*. With this refinement not only clicked links are considered for the history but also links which only have been displayed to user. This is under the assumption that a link not followed will be less interesting in the future. By making the presentation of such links more inconspicuous, the cognitive burden on the users is reduced. A prototype implementation is shown for news sites, where not following a link the first time usually means that the link will not be followed in the future as well.

Index Terms

World Wide Web, Browsers, User interfaces

1. Introduction

The World Wide Web has an amazing history and it is the most successful hypertext system in use today. Its usage scenarios encompass distributing information, augmenting social interaction, providing services (computational, storage, etc.) and many more[1]. The main work horse when accessing content delivered over the web is a web browser, be it one by Apple, Google, Microsoft, Mozilla, or some other party. Many of the concepts and features available in any modern web browser have their origin in hypertext research. One of those concepts is the concept of a history of visited hypertext nodes, or pages in the case of the web. This feature is implemented as browsing history in nearly every browser available and is one of the oldest features of browsers (it has already been available in the Mosaic browser (Andreessen, 1993)). Surprisingly, this feature can still be refined.

1.1. Problem Specification

To identify such a possible refinement let us take a look at the scenarios of reading newspapers. Although newspapers are (to some extent) replaced by their on-line versions (which are read on tablets, smart-phones, etc.), the printed, offline version of a newspaper still provides major benefits for reading. Let us take a look at the typical usage scenarios, offline as well as online.

1. To be precise: Services are often provided by the underlying Internet, the web providing the user interface to those services. Four our purpose it is sufficient to only talk about the web.

Figure 1. A screenshot of the online version of *The New York Times*

1.1.1. Offline Newspaper Reading. The following steps describe a typical offline newspaper reading session.

- The reader has the newspaper delivered and at (or in) hand.
- The reader skims through the headlines of the newspaper.
  - If a headline is interesting, the reader reads the article.
  - After that the reader resumes skimming.
  - This repeats for some time.
- The reader finishes skimming or reading and puts the newspaper down.

This scenario, of course, excludes any kind of disturbance, be it delivered by a smartphone or by kids or anything else.

1.1.2. Online Newspaper Reading. Now let us take a look at the newspaper reading scenario when the reader is reading the same newspaper on the web as provided, for example, by *The New York Times* on their online presence (see Figure 1)[2].

The previously described steps now look as follows:

- The reader loads the newspaper site into his browser. An overview page is usually displayed.
- The reader skims through the headlines of the newspaper.
  - If a headline/teaser is interesting, the reader clicks on the provided link.
  - The article is either loaded into the current tab or in a new tab (depending on the type of click by the reader).
  - The reader reads the article.

2. *The New York Times* (http://www.nytimes.com/) was chosen as example throughout this paper not because there is any affiliation with the newspaper but because of it being a well known English newspaper.

Figure 2. *The New York Times* revisited, with content slightly changed from Fig. 1.

  - He then navigates back to the overview page (either by going back in the browser history or by closing the tab).
  - The reader resumes skimming.
  - This repeats for some time.
- The reader finishes his browsing session (e.g., by closing the tab or the browser).

So far the difference between the two activities is not that big. The main differences are the content delivery method and the media involved (paper versus some kind of screen). Other differences are attributed to the different metaphors used (like opening a tab). The scenarios start to diverge when the reader decides after some time to continue reading his newspaper.

In the offline case we can assume the following: First, the person knows (at least approximately) where he stopped reading and what he has read already. Secondly, and more important, the newspaper still has the same content[3].

In the online case the second assumption does not hold any more. During the passed time span (if long enough but usually within a day), newsworthy events have taken place in the world and the overview page of the site has changed: older article teasers are removed, newer are added (see Figure 2 for an updated page, screenshot taken after approx. 12 hours), and the position of unchanged teasers within the layout may have changed. The problem then for the reader is that this can happen anywhere on the page, as the page often is (and not only in this example) structured into different sections which may change differently. So the mental map of the reader what he has already read and what is yet new is outdated. The browser history is of limited use here: only the articles actually read by the user (and thus those which have been clicked on, resp., clicked on the links to follow) are coloured differently. Those article teasers which the

3. Except for some rare incidents, e.g., involving newspaper modifying pets.

reader has previously decided against to read are still displayed unchanged in their visualisation and the reader has to decide again if he wants to read the associated article.

So the reader now has to perform the following steps, repeatedly for each article teaser which is yet unvisited and thus not displayed differently:

a) The user reads the headline or description of the article teaser (In case of a site with links only, some system may give the user additional information about the link in the form of a preview picture (e.g., as described in (Kopetzky and Mühlhäuser, 1999)) when hovering with the mouse over the link). *b*) If that information is not sufficient, the reader reads the remainder of the teaser. *c*) The user then decides to follow the link or not (again), and *d*) finally clicks on the link or not, depending on the previously made decision. Only then a request is (probably) sent to the web server.[4]

This leads to the conclusion that the default history feature of current browsers is inadequate as it does not help the user in this case. It only stores the information about which links the user has clicked and followed (as designed), but not the decision which links the user decided not to click.

As decision making requires a mental effort which can even contribute to reduced self control (see (Vohs et al., 2008)), a software which requires less decisions to be made is better than one which requires more decision to be made in the light of this aspect.

There are already solutions for this problem available, although they do not address the problem directly, solving it more or less inadvertently: They are described in section 2 in the context of newspapers.

2. Related Work

2.1. Related to the Hypertext Aspect

The history of browsing dates back to the seminal paper "As we may think", albeit there called *trails*. In addition, trails are a more powerful concepts, as they include history information from other users (Bush, 1945). An overview of the different ways of navigating hypertexts is contained in (Jakob Nielsen, 1995).

The feature of managing the history in current browsers is basically the same as in early browsers. Only filtering by time or search term are new options (see Fig. 3, 4 and 5 for the history UI of three modern browsers).

2.2. Related to the Newspaper Aspect

Ihlström et al. propose eight design recommendations for online newspapers (Ihlström and Lundberg, 2004) which are based on a study of nine Swedish online newspapers. Some of the studied newspapers operated with time stamps which were used to show the freshness of the news. One newspaper decided to abstain from using time stamps as there were seen as only for internal use. Although timestamps are a proven method of marking newer elements, they still place overhead on the reader: The reader has to know exactly at what time he read the paper to decide if an article with a time stamp is new for him or not. If

4. *Probably* because the request action still depends on the result of the evaluation of the click-event, and the evaluation can be modified by code.

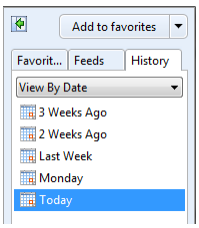

Figure 3. Internet Explorer history.

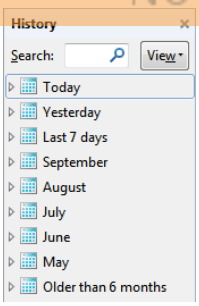

Figure 4. Firefox history.

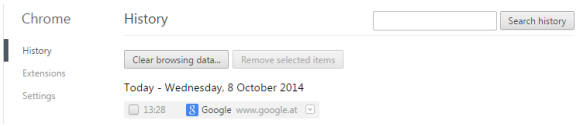

Figure 5. Chrome history

in doubt, he has to read more information about the article to decide. *The New York Times* has timestamps on the frontpage on some of its articles, but they are in small print and can easily be overseen.

Bucy defines the category of immediacy items, which contains date or time-stamped news stories; news ticker with current headlines; indication of new content; date or time of last update (Bucy, 2004). The history of the seen, although oriented to the past, belongs to this category as well, as it stresses new content.

Some newspapers avoid this problem by publishing a digital paper version of their offline product, for example the e-Edition for *The New York Post*[5] or the epaper version for *Der Standard* (www, 2014c). One of the advantages of these products is that they look exactly as the printed product, enhanced with some feature possible only online (zooming into an article, clicking on links for faster navigation). On the other hand, they have some of the same disadvantages as the printed version, e.g., they are only as current as the printed product and breaking news can not be included in them.

3. A Possible Approach

Before a solution will be presented, let us take a look at the different structures of news sites. There are I) sites which contain mainly links on the overview page (e.g., the heise newsticker siteheise), or II) sites with only some articles with teasers and the rest links on the overview page (e.g. the ORF site (www, 2014b), which is special in the way that the teasers are only images, see Fig. 8). Then there are III) sites which mainly contain teasers and only few links alone, as already seen with *The New York Times*. Let us define these different sites as type I, II, and III, resp., as handling of these types requires different methods.

Figure 6 shows *The New York Times* how the browser should display the site with the envisioned solution realized: Older articles and teasers are made transparent and new ones are easy to find.

5. http://nypost.newspaperdirect.com/epaper/viewer.aspx

Figure 6. *The New York Times* revisited and displayed as envisioned, with new content available.

Figure 7. *The New York Times* revisited and displayed as envisioned, no new content available.

As this was made using the implemented prototype, a limitation of the prototype can be seen here already: The Watching box is unaffected. More about this in Section 5.1.

Figure 7 shows *The New York Times* how it looks like if no new content has been added to the page (besides the limitation concerning the watchbox).

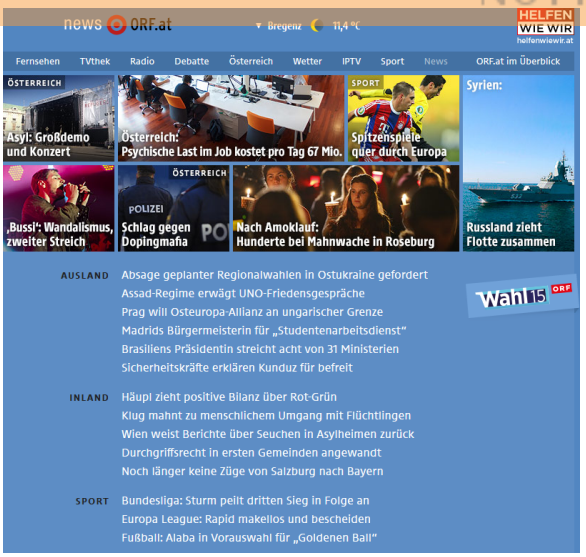

Figure 8. Screenhot of ORF site, images are teasers

Additionally, some definitions of terms are needed.

Article an piece of text in an online newspaper. Usually this is on its own page an is linked from the overview page.

Teaser a short piece of text, sort of abstract of an article, usually on an overview or title page (the articles in Fig. 1 on the overview page of *The New York Times* are teasers).

Overview Page The overview page usually contains only teasers or some teasers and lots of links (as on the ORF online site (www, 2014b)). The overview page is usually loaded first when reading an online newspaper.

Link a standard hypertext link.

Visited link a link the reader already has visited (or at least clicked on so that the browser already has loaded the link).

As the history feature of browsers only operates with links, the focus is first on extending the history paradigm by defining a *seen link* l_s. A *seen link* is defined as a link the browser has loaded and displayed to the user. The assumption behind this definition is that the reader has seen the link when it is displayed, as there is currently no other way to learn that information[6]. In addition, a *displayed link* l_d is currently displayed to the user.

Based on this definition, the following four sets are defined: all links ($\mathcal{L}$), all links loaded and not displayed yet ($\mathcal{L}_\mathcal{N}$), all links already loaded and displayed at least once (and thus being seen, ($\mathcal{L}_\mathcal{S}$), and all links navigated by the user ($\mathcal{L}_\mathcal{H}$) - this set is also known as the history. The following relation holds for all practical reasons $\mathcal{L} \supset \mathcal{L}_\mathcal{N} \supset \mathcal{L}_\mathcal{S}$. The intersection of each of the previous three sets with the history set $\mathcal{L}_\mathcal{H}$ may or may not be empty.

Let us assume also that the history is ever growing and no links are removed from the history after some time, either automatically or manually.

In addition, for a working approach, the following requirements need to be met:

1) A way to determine what constitutes an article.
2) A way to uniquely identify an article.

6. One could use eye tracking technology, but this requires additional hardware.

Both requirements are non-trivial to fulfil. Although an overview page displays links and teasers in a structured layout, the HTML, CSS, and JavaScript code describing the content, the layout, and other aspects are different for each news site and do not operate with universally accepted definitions of articles and teasers. Also, articles and links are rarely given a unique ID. This is no problem for links, as links per definition uniquely identify the target, but identifying an article can only be done using heuristics depending on the structure of the site containing the article.

Based on this definitions and the definition of site types in section 3, a first approach for type I sites can be given.

3.1. Approach for Type I Sites

As type I sites mainly contain links, the definition of a link and its property, that a link uniquely identifies a target, can be used for realising a history of the seen links. For those sites the link represents the article. Thus iterating over all <*a*> tags of a page enables us to render such links differently.

Initially the set of the display links is empty:

$\mathcal{L}_\mathcal{S} := \{\}$

After loading each page the following algorithm processes the link visualisation:

Handling of links during a browsing session:
begin
  for $l \in$ {all hypertext links of the DOM} do
                                – standard history handling
    if $l \in \mathcal{L}_\mathcal{H}$ then
      showLinkAsVisited(l)
    fi
                                – handling of displayed links
    if $l \in \mathcal{L}_\mathcal{S}$ then
      rememberLinkAsDisplayed(l)
      $\mathcal{L}_\mathcal{N} := \mathcal{L}_\mathcal{N} \setminus \{l\}$
      $\mathcal{L}_\mathcal{S} := \mathcal{L}_\mathcal{S} \cup \{l\}$
    else
      showLinkAsDisplayed(l)
    fi
  od
end

Changing the visualisation of the link can be done with the following function:

proc showLinkAsDisplayed(link l) $\equiv$
  modify color/transparancy/etc. of l
end

3.2. Approach for Type II and III

For type II and III sites using the link alone is not sufficient. Although for most sites a teaser contains a link which is unique, we also want to mark the surrounding

Listing 1. Structure of an article of the *The New York Times*

```
<div class="collection">
        <article class="story theme-summary"
                id="topnews-100000001234567"
                data-story-id="100000001234567" ...>
                <h2 class="story-heading">
                        <a href="http://www.nytimes.com/...html">
                                Article Headline</a>
                </h2>
                <p class="byline">...</p>
                <p class="summary">Article summary.</p>
                ...
        </article>
</div>
```

Listing 2. Settings for *The New York Times*

```
url : ['nytimes\.com'],
        upTrigger: "../a",
        parentHints : [
                "ancestor::div[contains(concat(' ', @class, ' '),
                 ' collection ')]"
        ]
```

content of the link as seen. The surrounding context usually constitutes the teaser, possibly with a header.

The challenge is how to identify the context for a given HTML page.

One possible approach is to examine the HTML surrounding the location of the link. Most of the sites used in this paper employ a hierarchy of DIV elements, where the DIV element starting an article contains some unique class name. The DIV element with this unique class containing the link can then be used to identify an article.

To realize this new requirement, a change to the showLinkAsDisplayed function is sufficient:

proc showLinkAsDisplayed(link l) $\equiv$
  $elem =$ surrounding element of l as per site heuristic
  modify color/transparancy/etc. of $elem$
end

3.3. An Example

The listing 1 contains an HTML snippet from the *The New York Times*-page. It contains a teaser article with the link to the full article.

To identify a link and the surrounding article element, the configuration shown in listing 2 contains the heuristics for *The New York Times*.

The *url* parameter contains the url the settings are for. The *upTrigger* specifies the element we are looking for, in this case the *a*-tag, specified as XPATH-expression. Finally, the *parentHints* specifies an XPATH-expression for locating the surrounding element. This expression will be evaluated starting from the location of the *upTrigger*.

Other sites may have different heuristics.

4. Implementation

The prototypical implementation described here builds on the open source browser FireFox (in a current iteration) and a special plugin for this browser, GreaseMonkey[7].

GreaseMonkey allows for writing small JavaScript programs for specific web sites, called scripts henceforth. These scripts are then executed within a sandbox after a page of the specified web sites has been loaded. The scripts are allowed to access and manipulate the DOM of the currently loaded page. Another feature of GreaseMonkey is the ability to store data in a database via a simple interface. This database allows to store data between different browser sessions.

These scripts work without the overhead necessary for a full fledged plugin and are thus easier and faster to implement. Usually they are quite small as well, the typical use case being the compensation for some quirk of a web page or some small automation tasks.

4.1. Source

The current version of the source of the script can be found on GitHub (www, 2014a). An easy installation option is available at GreasyFork[8] or OpenUserJS[9].

5. A Better Mouse Trap

As the current implementation has some limitations, the following subsections will elaborate about those shortcomings and how to address them.

5.1. Limitations

The following list contains the known limitations of the current implementation.

- The prototype cannot handle dynamic loading of content after the page load event. This can be seen in Fig. 6, where the lower right box is unaffected.
- Two additional pieces of software need to be installed in order to provide the new history: GreaseMonkey and the script. Thus there are two more potential security problems and two more pieces of software to distribute and update (although that happens nearly automatically).
- Performance is not optimal, first, because it is written in JavaScript, and second, because the interface to the database provided by GreaseMonkey is very primitive.
- A more serious limitation is that script relies on heuristics to implement the effect for different sites. If a site decides to do a restructuring of their structure, the heuristic for the specific site has to be adapted and distributed.
- If you want the feature for a new site, a new heuristic may have to be devised.

7. http://www.greasespot.net/
8. https://greasyfork.org/scripts/4273-history-of-the-seen
9. https://openuserjs.org/scripts/theoky/History_of_the_Seen

To address these shortcomings there are different options which are listed as follows:*a*) Implement a plugin for the browser. *b*) Change the browser itself. *c*) Adapt the server. *d*) Extend HTML. Option a) is only slightly better than the current situation: there is only one piece of software to be distributed. The other disadvantages still remain. Option b) is a little bit better than option a), but still has the shortcoming of employing a heuristic. So new heuristics can only be deployed alongside browser updates. Option c) has the downside that the server needs to manage an information which is only generated on the client, and for all clients.

Only option d) can provide a lasting solution to all mentioned shortcomings, as the responsibility of defining which element of a page has to be used for the seen history can be transferred to the location where the HTML - and thus the page - is generated: to the server-side. Only there it can be easily defined which elements should be treated in this way by providing a special annotation. Changes can happen there without influencing the client (or clients) any more. The downside of these options is, that there are initially changes in many places: within the standards, within browsers interpreting the standard, and within the servers providing content, as they need to be changed to provide the necessary annotation.

6. Conclusions

This paper shows that the history feature of modern browsers can still be refined and presents the concept of *history of the seen* as refinement.

References

History of the seen at Github. https://github.com/theoky/HistoryOfTheSeen, October 2014a.

ORF. http://www.orf.at, October 2014b.

Der Standard, ePaper. http://epaper.derstandarddigital.at/, October 2014c.

Marc Andreessen. NCSA mosaic technical summary. Technical report, National Center for Supercomputing Applications, May 1993.

Erik P. Bucy. Second generation net news: Interactivity and information accessibility in the online environment. *International Journal on Media Management*, 6(1-2):102–113, 2004. URL http://www.tandfonline.com/doi/abs/10.1080/14241277.2004.9669386.

Vannevar Bush. As we may think. *The Atlantic*, 176(1):101–108, July 1945. ISSN 1072-7825.

Carina Ihlström and Jonas Lundberg. A genre perspective on online newspaper front page design. *Journal of Web Engineering*, 3:50–74, 2004. URL https://130.236.177.26/~TDDC62/material/JWE050-074.pdf.

Jakob Nielsen. *Multimedia and Hypertext: The Internet and Beyond*. Morgan Kaufmann, March 1995. ISBN 0125184085.

Theodorich Kopetzky and Max Mühlhäuser. Visual preview for link traversal on the world wide web. *Computer Networks*, 31(11-16):1525–1532, 1999.

Kathleen D. Vohs, Roy F. Baumeister, Brandon J. Schmeichel, Jean M. Twenge, Noelle M. Nelson, and Dianne M. Tice. Making choices impairs subsequent self-control: A limited-resource account of decision making, self-regulation,

and active initiative. *Journal of Personality and Social Psychology*, 94(5):883–898, 2008. ISSN 1939-1315, 0022-3514. doi: 10.1037/0022-3514.94.5.883. URL http://doi.apa.org/getdoi.cfm?doi=10.1037/0022-3514.94.5.883.

Appendix

The following web sites have been used throughout this paper:

heise.de a German IT news site (http://www.heise.de/newsticker/).

The New York Times an English newspaper (http://www.nytimes.com/).

orf.at an Austrian news site (http://news.orf.at/).