Data Cleaning Methods for Client and Proxy Logs

Harald Weinreich
VSIS, Department of Informatics
University of Hamburg, Germany
harald@weinreichs.de

Hartmut Obendorf
ASI, Department of Informatics
University of Hamburg, Germany
hartmut@obendorf.de

Eelco Herder
Department of Computer Science
University of Twente, The Netherlands
herder@cs.utwente.nl

ABSTRACT

In this paper we present our experiences with the cleaning of Web client and proxy usage logs, based on a long-term browsing study with 25 participants. A detailed clickstream log, recorded using a Web intermediary, was combined with a second log of user interface actions, which was captured by a modified Firefox browser for a subset of the participants. The consolidated data from both records revealed many page requests that were *not directly* related to user actions. For participants who had no ad-filtering system installed, these *artifacts* made up one third of all transferred Web pages. Three major reasons could be identified: *HTML Frames* and *iFrames*, advertisements, and automatic page reloads. The experiences made during the data cleaning process might help other researchers to choose adequate filtering methods for their data.

Categories and Subject Descriptors

H.5.4 Hypertext/Hypermedia: User issues.

General Terms

Experimentation, Human Factors

Keywords

Clickstream study, proxy log analysis, data cleaning

1. INTRODUCTION

In Winter 2004/2005, we conducted a long-term Web usage study with 25 participants from Germany and the Netherlands. The logging environment for the study consisted of two complementing tools: all participants made use of an intermediary system [1] filtering all `http` traffic. The system added JavaScript events and processing functions to every Web page. This code 'instrumented' the Web pages to record user clicks, form entries and various other browser parameters. The data were sent to the intermediary using dedicated `http` requests[1]. Our logging tool was based on the Scone framework [4; 8] and IBM's WBI system [5].

The result was a rich 'intermediary log' with details about every selected link anchor, click positions, load and stay times, and all submitted form data. The current state of the history of the browser window permits identifying backtracking events. Moreover, the system provided unique window and frame names to distinguish the area of page load. The intermediary analyzed all transferred documents and recorded descriptive data about their contents, size and hyperlink structure.

In addition, 15 of the 25 participants agreed on using an instrumented version of the Firefox 1.0 browser for the time of the study. We altered its XUL[2] (XML User Interface Language) code to record the interaction with its user interface widgets, as well as window and tab operations and link clicks. This data listed precisely all user actions with the browser itself, but missed information on the pages visited and the interaction of the participants with the documents, like entered form data and click positions.

Using exact timestamps, the user interface log could be merged with the page request log of the intermediary to gain more detailed and accurate data. Due to the experimental setup, we had to cope with ten intermediary log streams without browser UI data.

The first analyses of the recorded data showed that serious data preprocessing would be necessary to obtain a 'clean' log of navigation actions: many of the page requests recorded by the intermediary were *not directly* related to user actions. For our purposes – analyzing user behavior – these events had to be classified as 'artifacts', as they were not connected with a purposeful action, and often not even evident to our participants.

Although 'data cleaning' and 'ETL' (*Extract, Transform, Load*), are quite extensively explored fields of data processing, and many tools and techniques exist, the specific research area of cleaning Web logs seems to be quite unexplored: only few publications and no guidelines exist that mention the issues we experienced while cleaning the data of our Web client and proxy logs to extract the desired user actions.

We identified several categories of page requests that evidently were not related to deliberate user actions: HTML Frames, iFrames, advertisements and automatic page reloads were the main factors; automatic page redirects (using JavaScript or the 'redirect' meta element of HTML) accounted for a less significant amount of log entries (see Figure 1). The data in our Web activity logs were too extensive for manual data cleaning, so we developed several heuristics and methods to isolate page loads not directly related to intended user actions.

Based on these heuristics, several Python scripts were implemented that processed all log files into an SQL database and filtered the recorded user actions in three steps: first, most of the artifacts that could be identified by URL or by other, more immediate means, were removed; second, the data streams from the Web client and proxy logs were combined; finally, the proxy

[1] The image object was used to send http 'get' requests to the intermediary as soon as specific data had been collected, like the completion of a page load or the click on a link. The intermediary returned an empty dummy image, which was not displayed. These data transfers were unnoticed by the users. Alternatively, the `XMLHttpRequest` object could have been used.

[2] The User interface of Mozilla and Firefox browsers are written in XUL, a language based on XML and JavaScript: http://www.mozilla.org/projects/xul/

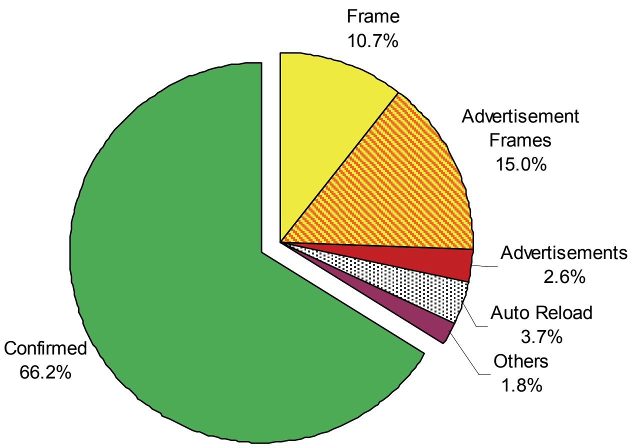

Figure 1: Proportions of artifacts for users w/o ad blockers

events that were not matched by client events were postprocessed. Several iterations for the development of the algorithms and the software were required.

Data cleaning was a *very* laborious process, due to the vast number and the complexity of recorded events: we recorded more than 160,000 page requests with the intermediaries and nearly 150,000 user interface actions with Firefox[3], and every entry had up to 27 parameters. It took dozens of hours to sight the data, to develop rules how to distinguish 'real' user requests from artifacts and to verify the results of our data processing tools.

The following sub-sections address the main methods and heuristics developed during this process.

1.1 HTML Frames and iFrames

HTML framesets break the document metaphor: what is visible for the user does not originate from a single HTML document and many 'unusable' events are created as every subframe creates a page request to a different HTML file. As an example, a request to the (invisdible) top frame of figure 2 would automatically be followed by requests for the two sub frames. Together these three requests constitute one page view and correspond to one user action. Therefore, a merging of these requests seemed to be necessary.

We identified frames by the 'name' of the frame (which was obtained from the browser object `window.name)` and the name of the parent frame `(window.parent.name)`. These attributes were read by the embedded JavaScript code and transmitted to the intermediary.

However, the interpretation of matching frame events did not only require comparing frame and window names, it was also necessary to introduce *timeouts*, as user actions like link clicks can lead to the request of one or more pages within a frameset (for example if JavaScript is used). The required timeout for this merging process depended on the connection speed of the user. For our data set, timeouts between 1.8 seconds (leased line) and 3 seconds (isdn) provided the best results.

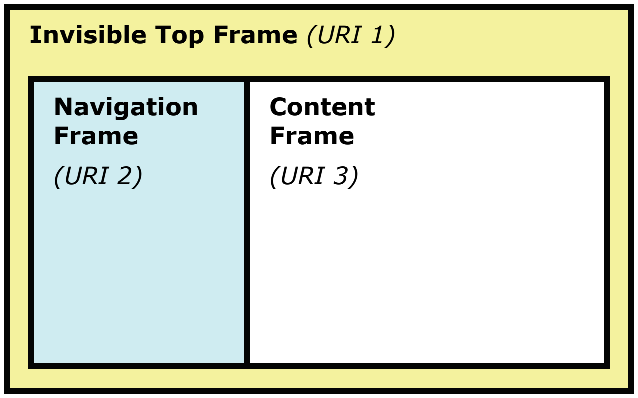

Figure 2: At least three http requests are involved if a Web page uses HTML frames

Nevertheless, one problem remained: frame events are ambiguous, as several URIs are involved in one user action. The decision, which of the page requests is the most relevant, depends on the way the data will be interpreted or analyzed. An example will clarify the dilemma (see figure 2): if a user selects a link in the left *navigation frame* and a page is loaded in the right *content frame*, the action occurs in the left page but the reaction affects the right page with another URI. Each of these three URIs (including the parent frame) could be regarded as *principal address* for the user action in the clickstream log.

The chosen URI mapping affects observed load times [6], the visited pages [3; 7], average document sizes as well as the relative mouse-cursor position. Therefore, we decided to exclude frame pages for some analyses [see 9], like the average load times, the revisitation rate and the link click positions (as they were relative positions within the documents and not absolute screen positions).

Another large proportion of non-user-initiated page requests in the intermediary log was caused by iFrames ('Inline Frames'). This HTML element allows embedding other Web pages in a specified area of a Web document. Every iFrame creates a separate page request without any extra user activity. Our participants visited many commercial pages that made extensive use of iFrames (e.g. eBay and some news sites), which were mainly used to dynamically include advertisements from dedicated servers (see following section). We therefore excluded inline frame requests for most analyses; this changed some results: e.g. it significantly decreased the revisitation rate.

1.2 Advertisements

Page requests caused by online advertisement turned out to be another significant source of noise. Apart from the mentioned iFrames, ads appear quite frequently in (automatic) pop-up windows. Such JavaScript-initiated windows are often not the result of a deliberate user action and the offered links were almost never followed. Furthermore, most advertisements appeared on Web pages with a high popularity and were visited frequently, such as popular news and commerce sites. Consequently, the advertisement pages were also revisited; this increased the overall revisitation rate, although the users only intended to revisit the main page.

Furthermore, about two-thirds of our participants had pop-up and ad blockers installed. As we wanted to base our calculations on a consistent data pool, we decided to exclude these page requests

[3] This included also many actions that did not lead to a page request, like the selection of browser tabs or closing the browser window.

for all users of our study. Consequently, we needed a way to identify all automatic page requests related to merchandising.

Screening the intermediary log data and comparing it with the user interface action log, we could validate three techniques for identifying 'pure' advertisement HTML pages.

The first method considers the value of the `window.name` object of an iFrame and its main window. Partly names like '`LVM Frame`', '`merchandizing*`'[4] or '`*_ads_frame`' are used for advertisement inline frames. In addition, the ad content is frequently requested from a dedicated server with another domain name as the main page, and therefore JavaScript access from the promotion page to the objects of the parent document is blocked by browser security policy. The error message caught by our embedded JavaScript code (`PARENT_FRAME_ACCESS_DENIED`) was thus another indication for such inline frames. However, some of these errors were not related to advertisements, as normal frames can cause them as well and thus this message mainly helped to improve the other heuristics.

The second technique is based on a list of over 10,000 domain names with Web servers that provide only advertisements, like `ad.doubleclick.com` or `*.ivwbox.de`. This list was taken from the *SupertrickXG* Web site[5], a project to register 'malware, porn and adware servers'. Although this list is extensive, we had to add some additional national advertisement servers, based on the prevalence in the Web log files.

The third method analyses the path and filename in the URI. We created a set of regular expressions to identify page loads that are caused by pure advertisement pages. These expressions were not only based on our manual examination of the log data, but we also considered lists of regular expressions, created in conjunction with the *Mozilla Adblock Project*[6]. Two examples for such expressions are:

```
[\W\d](ad(banner|click|-?flow|frame|ima?g(es?)?|
serv(er|e)?|_string|vertisements?|v|vert))[\W\d]
```

and

```
[\W\d]click(stream|thrutraffic|thru|xchange)[\W\d]
```

However, it should be noted that extreme care has to be taken when assembling these filters, as overly simple heuristics will incorrectly identify 'real' Web pages as ads. An example is `www.`**`ad`**`.nl`, a popular Dutch news site with a very 'suspicious' domain name.

Extensive data auditing verified that these three techniques together did identify most unwanted page loads[7]. For the group of participants that did not apply any kind of ad-blocker (8 users) frame and advertisement artifacts represented over 28 percent of page requests (figure 1), for the remaining users – making use of advertisement and pop-up blockers –, such events were rather marginal (below 1%). The proportion of pure advertisement page requests seems remarkably high, as it does not consider online promotion realized as pure text, embedded images, or flash animations.

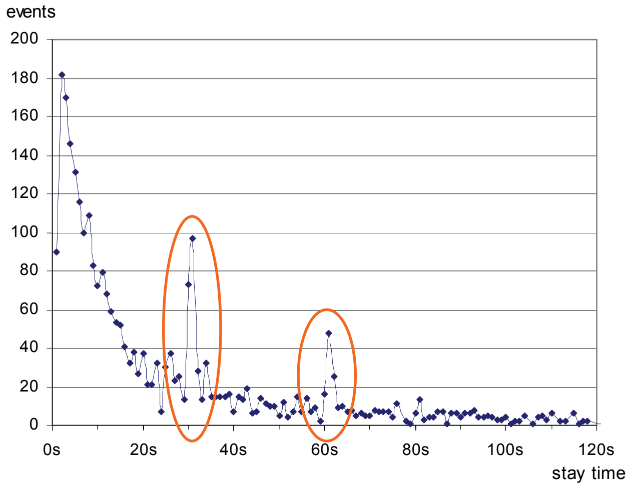

Figure 3: Stay time distribution for one of the participants

1.3 Automatic Page Reloads

A third source of non-user-initiated page requests was the *automatic* reloading of pages. We identified several reasons for Web page reloads not initiated by our participants. Either embedded JavaScript code like

```
window.location.reload();
```

triggered by a timer, or the 'refresh' meta-element

```
<meta http-equiv="refresh" content="2">
```

was often used for this purpose. In other cases, external applications, like instant messaging agents, were responsible for the artifacts. All these methods produce a constant time interval between reloads that we used to identify these events.

The auto reloads became apparent when we analyzed the individual page revisitation rate and the page stay time distribution. Automatically reloaded pages emerged as frequent revisits to one URI in a row with a repetitive stay time. As a result, they appeared as peaks in the stay time distribution of certain users (see figure 3) and we could identify many of these entries manually. Furthermore, we implemented an algorithm to detect automatic page reloads: If three more requests to the same URI occurred consecutively in the same window and they all had approximately the same stay time (plus/minus 1 second), these page requests were marked as artifacts as well.

Overall, *automatic* page reloads contributed nearly four percent of the page requests. However, the ratio differed severely between individuals: some participants did not visit any periodically reloaded pages, while for others they made up over 20% of all transmitted pages. The relation is influenced by the kind of Web sites visited, the applications used and the behavior of the user: by leaving a browser window opened in the background of her desktop, or by keeping the computer turned on overnight, a significant amount of irrelevant events was created for our study.

In contrast to frames and advertisements, automatic page reloads involve pages that have originally been explicitly requested by the user. Some sites use this technique to update the page contents, e.g. to refresh the headlines of a new portal page. We decided to exclude these requests, as they were not user actions, only few sites use this technique and it biased the stay times of some users significantly.

[4] The asterisk represents a wildcard character.

[5] The 'Supertrick XG Project' home page can be found at: http://www.filesharingplace.com/supertrickxg/main.htm

[6] The 'Adblock Project' is located at http://adblock.mozdev.org/

[7] At the same time these results can be used to improve current ad-blocking practice.

2. CONCLUSION

As became clear during the first analyses of the comprehensive datasets we had gathered, data cleaning and confirmation of user-initiated events were crucial to be able to relate recorded events to user actions. Previous studies did not make use of similar data consolidation methods, probably because the amount of such 'noise' was lower in the past: in 1995, advertisements were still hardly known on the Web, and Bruce McKenzie [3] told us that even in 2000 the effect of such requests could be neglected. Furthermore, iFrames were first introduced in MS Internet Explorer 3 (1996), and HTML 4.0 (first released Dec. 1997), but it took several years before they became popular, as the Netscape Navigator 4 (used in McKenzie's study [3]) did *not* support them.

Using the techniques presented, we were able to confirm 137,272 user-initiated *navigation actions* of our participants. They visited 65,643 distinct URIs and 11,928 different domains. The personal Web usage varied widely in browsing style and activity: the participants visited 19.5 to 204.8 pages per active day (every day in which at least one event was logged). A first analysis of the data is presented in [9].

Still, even after the data cleaning process, frames continued to be a nuisance for the interpretation of navigation actions: clicks on hyperlinks in one frame can lead to actions in one or several other frames. This makes it hard to define what exactly a page revisit is and when or how they occur.

The effects of and the necessity for data cleaning seem to be very dependent on the technical implementation of Web sites visited during a study. From our experience we can recommend to log all events (including possible artifacts), containing exact time stamps, URIs, and window and frame names. Instead of integrating data cleaning methods in the logging environment, the data should be filtered after finishing the study as many problems and effects can hardly be foreseen.

3. ACKNOWLEDGEMENTS

Many thanks go to Matthias Mayer (University of Hamburg), who contributed significantly to the study and the data cleaning process.

4. REFERENCES

[1] Barrett, R.; Maglio, P.P.; Kellem, D.C.: How to Personalize the Web. Proc. of CHI '97, Atlanta, GA, 1997.

[2] Catledge L.D.; Pitkow J.E.: Characterizing browsing strategies in the World-Wide Web. Computer Networks and ISDN Systems, (27)6, 1995, pp. 1065-1073.

[3] Cockburn, A.; McKenzie, B.: What Do Web Users Do? An Empirical Analysis of Web Use. Int. Journal of Human-Computer Studies, 54(6), 2001, pp. 903-922.

[4] Obendorf, H.; Weinreich, H.; Hass, T.: Automatic Support for Web User Studies with SCONE and TEA. Ext. Abstracts of CHI '04, Vienna, 2004, pp. 1135-1138.

[5] Maglio, P.; Barrett, R.: Intermediaries Personalize Information Streams. Communications of the ACM, 43(8), 2000, 96-101.

[6] Rajamony, R.; Elnozahy, M.: Measuring Client-Perceived Response Times on the WWW. 3rd USENIX Symposium on Internet Technologies and System, (SF), 2001.

[7] Tauscher, L.; Greenberg, S.: How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems. Int. J. of Human Computer Studies, 47(1), 1997, pp. 97-138.

[8] Weinreich, H.; Buchmann, V.; Lamersdorf, W.: Scone: Ein Framework zur evaluativen Realisierung von Erweiterungen des Webs. KiVS '03, Springer, 2003, pp. 31-42. http://www.scone.de/

[9] Weinreich, H.; Obendorf, H.; Herder, H.; Mayer, M.: Off the Beaten Tracks: Exploring Three Aspects of Web Navigation. Proceedings of WWW 2006 Conference, Edinburgh, UK, 2006.