WebQuilt: A Proxy-based Approach to Remote Web Usability Testing

JASON I. HONG, JEFFREY HEER, SARAH WATERSON, and JAMES A. LANDAY
University of California at Berkeley

WebQuilt is a web logging and visualization system that helps web design teams run usability tests (both local and remote) and analyze the collected data. Logging is done through a proxy, overcoming many of the problems with server-side and client-side logging. Captured usage traces can be aggregated and visualized in a zooming interface that shows the web pages people viewed. The visualization also shows the most common paths taken through the web site for a given task, as well as the optimal path for that task, as designated by the designer. This paper discusses the architecture of WebQuilt and describes how it can be extended for new kinds of analyses and visualizations.

Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: User/Machine Systems—*Human factors*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Evaluation/methodology*; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia—*User issues*

General Terms: Measurement, Design, Experimentation, Human Factors

Additional Key Words and Phrases: Usability evalution, log file analysis, web visualization, web proxy, WebQuilt

1. INTRODUCTION

About half the effort in developing and testing software applications is put into the user interface [Myers and Rosson 1992]. Unfortunately, successful user interfaces, even for simple systems, are difficult to create [Norman 1990]. For the web, the interface is even more challenging to get right, as evidenced by many studies showing that users have trouble completing common tasks online. Usability testing is necessary to help find and eliminate these problems; however, traditional testing techniques are inadequate. We have developed a system to assist web usability testing.

Authors' address: Group for User Interface Research, Computer Science Division, University of California at Berkeley, Berkeley, CA 94720-1776; email: {jasonh, waterson, landay}@cs.berkeley.edu; jheer@hkn.eecs.berkeley.edu.

1.1 Background

There is good evidence that web site usability remains a serious problem today. According to a 1998 survey of 239 experienced web users, 27% said that they found it somewhat or extremely difficult to find the product they wanted to buy [ZonaResearch 1999]. Furthermore, 62% said that they had given up looking for a product at least once in the past 60 days. Another study of fifteen large commercial sites showed that users could find the information they wanted only 42% of the time [Spool et al. 1998]. The usability of a web site is key to its success, particularly for increasing sales and repeat business on an e-commerce site [Tedeschi 1999].

The key to increasing the usability of a web site is in improving its design, by making the overall structure of the web site, the navigation and flow between pages, and the layout and wording on individual pages both coherent and meaningful for end-users. There are two problems all web design teams face in doing this: understanding what tasks people are trying to accomplish, and figuring out what difficulties people encounter in completing these tasks. Knowing only one or the other is insufficient. For example, a design team could know that someone wants to find and purchase gifts, but this is not useful unless they also know the problems preventing them from completing the task. Likewise, a design team could know that this person left the site at the checkout page, but this is not meaningful unless the design team also knows that he or she truly intended to buy something and is not simply browsing or doing a price comparison. There are a variety of methods for discovering what people want to do on a web site, such as structured interviews, ethnographic observations, focus groups, and questionnaires (for example, see Beyer and Holtzblatt [1998] and Rubin [1994]).

We concentrate on techniques designers can use for tackling the other problem, which is to understand what obstacles people are facing on a web site in the context of a specific task. Traditionally, this kind of information is gathered by running usability tests on a web site. A usability specialist recruits several participants representative of the target audience. These participants are brought to a usability lab and asked to complete a few predefined but realistic tasks. The usability engineer observes what stumbling blocks people come across and follows up with a survey and an interview to gain more insight into the issues.

One drawback to this type of usability test is that it is very time consuming to run with large numbers of people. It takes a considerable amount of work to schedule participants, observe them, and analyze the results. Consequently, the data tends to reflect only a few people and is mostly qualitative. These small numbers make it hard to cover all of the possible tasks on a site. Furthermore, small samples are less convincing when asking management to make potentially expensive changes to a site. Finally, a small set of participants may not find the majority of usability problems. Despite previous claims that about five participants are enough to find the majority of usability problems [Virzi 1992; Nielsen and Landauer 1993], a recent study by Spool and Schroeder suggests that this number may be nowhere near enough [Spool and Schroeder 2001].

Better tools and techniques are needed to scale up the number of participants and the number of tasks that can be managed for a usability test.

In contrast to traditional usability testing, server log analysis (see Figure 1) is one way of quantitatively understanding what large numbers of people are doing on a web site. It is easy to collect data for hundreds or even thousands of people with this approach since nearly every web server automatically records page requests. Server logging also has the advantage of letting test participants work remotely in their own environments—instead of coming to a single place, usability test participants can evaluate a web site from any location on their own time, using their own equipment and network connection. There are also many tools for server log analysis, with over 90 research, commercial, and freeware tools currently available on line.[1]

From the perspective of the web design team, however, there are some problems with server logs. Interpreting the actions of an individual user is extremely difficult, as pointed out by Etgen and Cantor [1999] and by Davison [1999]. Web caches, both client browser caches and Intranet or ISP caches, can intercept requests for web pages. If the requested page is in the cache then the request will never reach the server and is thus not logged. Multiple people can also share the same IP address, making it difficult to distinguish who is requesting what pages (for example, America Online, the United States' largest ISP, does this). Dynamically assigned IP addresses, where a computer's IP address changes every time it connects to the Internet, can also make it quite difficult to determine what an individual user is doing since IP addresses are often used as identifiers. While researchers have found novel ways of extracting useful user path data from server logs on a statistical level [Pitkow and Pirolli 1999], the exact paths of individual users remain elusive.

Furthermore, with standard server logs, users' tasks and goals are highly ambiguous. Much of the information provided by server logs is not very helpful in terms of improving usability. For example, knowing that 90% of a web site's visitors came from a ".com" domain around 3PM every day does not give the web design team any hints about what problems there might be. There are few hints as to who the visitors are, why they are coming to the site, what tasks they were trying to accomplish, whether or not they were successful, why they left, and what was their overall experience on the web site.

Access to server logs is also restricted to the owners of the web server, making it difficult to analyze sub-sites that exist on a server. For example, a company may own a single web server with different subsites owned by separate divisions. This also makes it impractical to do a log file analysis of a competitor's web site. A competitive analysis is important in understanding what features people consider important, as well as learning what parts of your site are easy-to-use and which are not.

One alternative to gathering data on the server is to collect it on the client. Clients are instrumented with special software so that all usage transactions will be captured (see Figure 2). Clients can be modified either by running

[1]Access log analyzers http://www.uu.se/Software/Analyzers/Accessanalyzers.html.

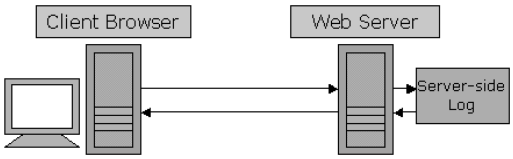

Fig. 1. Server-side logging is done on the web server, but the data is available only to the owners of the server.

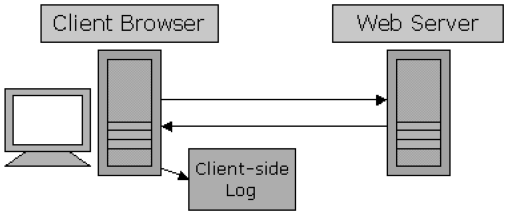

Fig. 2. Client-side logging is done on the client computer, but requires special software running in the background or having a special web browser.

software that transparently records user actions whenever the web browser is being used (as in Choo et al. [1998]), by modifying an existing web browser (as in Tauscher [1999] and Ergosoft [2001]), or by creating a custom web browser specifically for capturing usage information (as with Vividence [2000a]).

The advantage to client-side logging is that literally everything can be recorded, from low-level events such as keystrokes and mouse clicks to higher-level events such as page requests. All of this is valuable usability information. However, there are several drawbacks to client-side logging. First, special software must be installed on the client, which end-users may be unwilling or unable to do. This can severely limit the usability test participants to experienced users, which may not be representative of the target audience. Second, there needs to be some mechanism for sending the logged data back to the team that wants to collect the logs. Third, the software is platform dependent, meaning that the software only works for a specific operating system or specific browser. This limits the overall reach of the logging software. What is needed is a logging technique that is easy to deploy for any web site and is compatible with a number of operating systems and browsers.

1.2 WebQuilt

To better understand usability problems, designers need logging tools that can be used in conjunction with known tasks, as well as sophisticated methods for analyzing the logged data. Gathering web usability information is not a simple task with current tools. To recap, there are three things that could greatly streamline current practices in web usability evaluations:

1. A way of logging web usage that is fast and easy to deploy on *any* web site
2. A way of logging that is *compatible* with a range of operating systems and web browsers
3. Tools for *analyzing* and *visualizing* the captured data

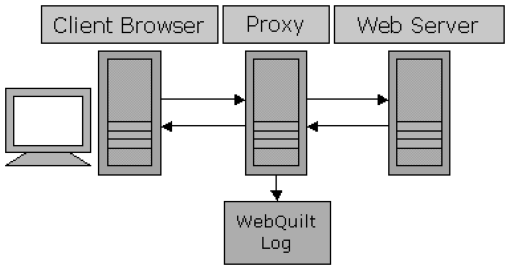

Fig. 3. Proxy-based logging is done on an intermediate computer, and avoids many of the deployment problems faced by client-side and server-side logging.

To address these needs, we developed WebQuilt, a tool for capturing, analyzing, and visualizing web usage. For the first and second needs, we developed a proxy-based approach to logging that is faster and easier to deploy than traditional log analysis techniques (see Figure 3). This proxy has better compatibility with existing operating systems and browsers and requires no downloads on the part of end-users. It will also be easier to make it compatible with future operating systems and browsers, such as those found on handheld devices and cellular phones.

To address the third need, we designed a visualization that takes the aggregated data from several test sessions and displays the web pages people viewed, as well as the paths they took. Through interviews with a number of web designers, we identified a few important indicators to look for when analyzing the results of a task-based usability test. These indicators include identifying the paths users take, recognizing and classifying the differences in browsing behavior, knowing key entry and exit pages, and understanding various time-based metrics (e.g., average time spent on a page, time to download, etc.). All of this data, when given the framework of a task and the means to analyze it, would help designers discover where users encounter obstacles such as confusing navigation, difficult to find links, and missing information.

Since we would not immediately have all of the solutions for analyzing the resulting data, we designed WebQuilt to be extensible enough so that new tools and visualizations could be implemented to help web designers understand the captured data. For example, WebQuilt can be used in conjunction with existing online survey and participant recruitment tools (such as those provided by NetRaker [2001] and Zoomerang [2001]). This allows designers to gather qualitative and demographic data about the site and users, and incorporate this information in the visualization/analysis.

WebQuilt is intended to be used as follows: A web designer would first set up several tasks. Next, he or she would recruit 20 to 100 participants to try out these tasks, and email them a starting URL that uses the WebQuilt proxy. As the participants go through the tasks, the proxy would automatically collect data. After all the data is collected, the web designer could use the WebQuilt tools to aggregate, visualize, and interact with the data to pinpoint usability problems. Once problems are found and fixed, the entire process is repeated in an iterative design process. This last point is especially important, as iteration

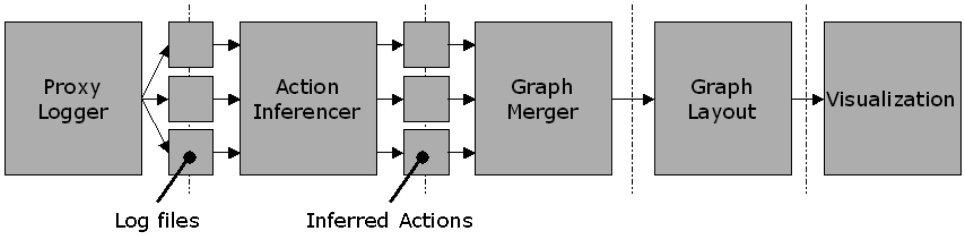

Fig. 4. WebQuilt dataflow overview. The proxy logger captures web sessions, generating one log file per session. Each log file is processed by the Action Inferencer, which converts the log of page transactions into a log of actions. The results are combined by the Graph Merger, laid out by the Graph Layout, and visualized by the Visualization component.

is widely considered to be a valuable technique for designing successful user interfaces [Gould and Lewis 1985].

In the remainder of this paper, we describe the architecture of WebQuilt and give a description of our current visualization tool. We then close with a discussion of related work and directions we plan to take in the future.

2. WEBQUILT ARCHITECTURE

WebQuilt is separated into five independent components: the *proxy logger*, *the action inferencer*, the *graph merger*, the *graph layout*, and the *visualization* (See Figure 4). The *proxy logger* mediates between the client browser and the web server and logs all communication between the two. The *action inferencer* takes a log file for a single session and converts it into a list of actions, such as "clicked on a link" or "hit the back button." The *graph merger* combines multiple lists of actions, aggregating what multiple people did on a web site, into a directed graph where the nodes represent web pages and the edges represent page requests. The *graph layout* component takes the combined graph of actions and assigns a location to each node. The *visualization* component takes the results from the graph layout component and provides an interactive display.

Each of these components was designed to be as independent of each other as possible. There is a minimal amount of communication between each component, to make it as easy as possible to replace components as better algorithms and techniques are developed. In the rest of this section, we describe each of these components in detail.

2.1 Proxy Logger

The goal of the *proxy logger* is to capture user actions on the web. As a proxy, it lies between clients and servers, with the assumption that clients will make all requests through the proxy. Traditionally, proxies are used for things like caching and firewalls. What WebQuilt does is to use a web proxy for usability purposes, with special features to make the logging more useful for usability analysis. In this section we first discuss how WebQuilt's proxy approach addresses the problems of current logging techniques, and continue with a description of the proxy's implementation.

2.1.1 *Advantages of Proxy Logging.* As discussed in the introduction, existing methods for capturing and generating web usage logs are not designed for gathering useful usability data [Etgen and Cantor 1999; Davison 1999; Pitkow and Pirolli 1999; Choo et al. 1998; Tauscher 1999; Ergosoft Laboratories 2001]. The proxy approach has three key advantages over the server-side approach. First, the proxy represents a separation of concerns. Any special modifications needed for tracking purposes can be done on the proxy, leaving the server to deal with just serving content. This makes it easier to deploy, as the server and its content do not have to be modified in any way.

Second, the proxy allows anyone to run usability tests on any web site, even if they do not own that web site. One can simply set up a proxy and ask testers to go through the proxy first. In order to do this, the proxy simply modifies the URL of the targeted site. End users do not have to change any settings to get started. This makes it easy to run and log usability tests on a competitor's site.

Finally, having testers go through a proxy allows web designers to "tag" and uniquely identify each test participant. This way designers can know *who* the tester was, *what* they were trying to do, and afterwards can ask them *how well* they thought the site supported them in accomplishing their task.

A proxy logger also has advantages over client-side logging. It does not require any special software on the client beyond a web browser, making it faster and much simpler to deploy. The proxy makes it easier to test a site with a wide variety of test participants, including novice users who may be unable or afraid to download special software. It is also more compatible with a wider range of operating systems and web browsers than a client-side logger would be, as it works by modifying the HTML in a platform-independent way. This permits testing with a more realistic sample of participants, devices, and browsers.

It is important to note that this approach is slightly different from traditional HTTP proxies. Traditional proxies (e.g., a corporate firewall) serve as a relay point for all of a user's web traffic, and the user's browser must be configured to send all requests through the proxy. The WebQuilt proxy differs in that it is URL-based—it redirects all links so that the URLs themselves point to the proxy, and the intended destination is encoded within the URL's query string. This approach is also used by web anonymizers (such as www.anonymizer.com). Using a URL-based proxy avoids the need for users to manually configure their browsers to route requests through the WebQuilt proxy, and so allows for the easy deployment of remote usability tests by simply providing the proper link.

2.1.2 *WebQuilt Proxy Logger Implementation.* The current WebQuilt proxy logger implementation uses Java Servlet technology. The heart of this component, though, is the log file format, since it is the log files that are processed by the *action inferencer* in the next step. To use the WebQuilt analysis tools, it actually does not matter what technologies are used for logging or whether the logger lies on the server, on a proxy, or on the client, as long as the log format is followed. Presently, the WebQuilt *proxy logger* creates one log file per test participant session.

Table I. Sample WebQuilt Log File in Tabular Format. The highlighted cells show where a person went back from the second requested page (Transaction ID 2) to the First (Transaction ID 1), and then forward again (Transaction ID 3)

Time	From TID	To TID	Parent ID	HTTP Response	Frame ID	Link ID	HTTP Method	URL + Query
15010	0	1	−1	200	−1	−1	GET	http://www.weather.com
28218	1	2	−1	200	−1	21	GET	http://www.weather.com/newscenter/topstories/010702severefortworth.html
32875	1	3	−1	200	−1	19	GET	http://www.weather.com/newscenter/topstories/010703typhoondurian.html
47975	3	4	−1	200	−1	25	GET	http://www.weather.com/newscenter/topstories/010703heatnevada.html
68772	4	5	−1	200	−1	27	POST	http://www.weather.com/local/94703 where = 94703 & what = & x = 8 & y = 14

WebQuilt Log File Format. Table I shows a sample log. The *Time* field is the time in milliseconds the page is first returned to a client, where 0 is the start time of the session. The *From TID* and the *To TID* fields are transaction identifiers. In WebQuilt, a transaction ID represents the Nth page that a person has requested. The From TID field represents the page that a person came from, and the To TID field represents the current page the person is at. The transaction ID numbers are used by the *action inferencer* for inferring when a person used the browser back button and where they went back. Table I shows an example of this, where a test participant went to a page, then back to the first page, and then forward again.

The *Parent ID* field specifies the frame parent of the current page. This number is the TID of the frameset to which the current page belongs, or −1 if the current page is not a frame. The *HTTP Response* field is just the response from the server, such as "200 ok" and "404 not found." The *Link ID* field specifies which link was clicked on according to the Document Object Model (DOM), and is useful for understanding which links people are following on a given page. In this representation, the first link in the HTML has link ID of 0, the second has link ID of 1, and so on. Both <A> and <AREA> tags are considered links. A value of −1 is used to denote that a link was not clicked on to get to the current page.

The *Frame ID* field indicates which frame in an enclosing frameset the current page is. These are numbered similarly to link IDs—a frame ID of 0 indicates the first frame in the frameset, and so on. If the current page is not a frame, a value of −1 is used. The *HTTP Method* field specifies which HTTP method was used to request the current page. Currently the proxy supports the GET and POST methods. The last fields are the *URL + Query* fields, which represent the current page the person is at and any query data (e.g., CGI parameters) that was sent along with the request.

The WebQuilt log format supports the same features that other log formats do. For example, the first row shows a start time of 15010 msec and the second row 28218 msec. This means that the person spent about 13 seconds on the page http://www.weather.com. However, it has two additional features other logging

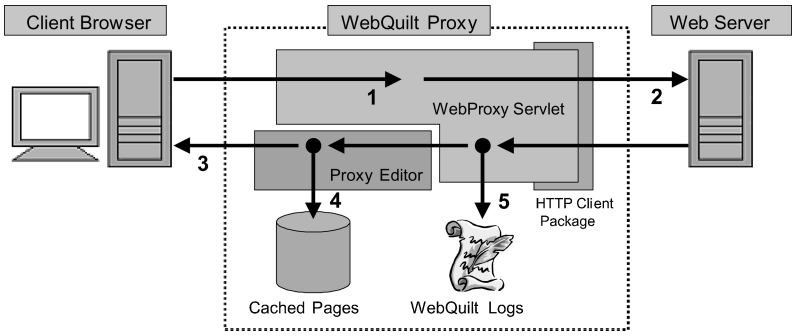

Fig. 5. Proxy architecture overview and sequence of operations. In step 1, the client request is processed. In step 2, the requested web page is retrieved. In step 3, all of the links are redirected to the proxy, and the web page is returned to the client. In step 4, the web page is cached, letting designers review the web pages traversed. In step 5, the entire transaction is logged.

tools and formats do not. The first is the Link ID. Without this information, it can be difficult to tell which link a person clicked on if there are redundant links to the same page, which is a common practice in web design. This can be important in understanding which links users are following and which are being ignored. The second is finding where a person used the back button. The highlighted cells in Table I show an example of where the person used the back button to go from transaction ID of 2 back to transaction ID of 1, and then forward again, this time to a different destination. Understanding this difference can help improve a site and reduce the ping-pong behavior commonly observed on the web.

WebQuilt Proxy Logger Architecture. Figure 5 illustrates the Proxy Logger's architecture. As mentioned before, the WebQuilt Proxy is built using Java Servlet technology. The central component of the system is the WebProxy servlet, which must be run within a Servlet and Java Server Page (JSP) engine (e.g., Jakarta Tomcat or IBM WebSphere). The Servlet engine provides most of the facilities for communicating with the client—it intercepts HTTP requests and hands them off to the WebProxy servlet, handles session management, and provides output streams for sending data back to the client. The WebProxy servlet processes clients' requests and performs caching and logging of page transactions. It is aided by the ProxyEditor module, which updates all the links in a document to point back to the proxy. Underlying the WebProxy servlet is the HTTPClient library [Tschalar 2000], an extended Java networking library providing full support for HTTP connections, including cookie handling.

Step (1)*—Processing Client Requests.* In the first step, an HTTP request is received from the client by the proxy. All WebQuilt specific parameters, such as the destination URL and the transaction ID, are extracted and saved. At this time the proxy collects most of the data that is saved in the log file. For example, the time elapsed since the beginning of the session is calculated, and other parameters such as transaction IDs, parent ID, and link ID are stored.

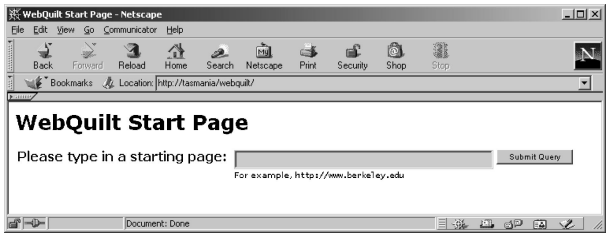

Fig. 6. Default page for the WebQuilt proxy. The proxy will retrieve and dynamically modify the URL that is entered.

There are two ways a person can start using the proxy. The first is by requesting the proxy's default web page and submitting a URL to the proxy (see Figure 6). This technique is mainly used for debugging. The second is by using a link to a proxied page. For example, suppose you wanted to run a usability study on Yahoo's web site. If the proxy's URL was:

http://tasmania.cs.berkeley.edu/webquilt,

then participants could just use the following link:

http://tasmania.cs.berkeley.edu/webquilt/webproxy?replace=
http://www.yahoo.com.

This method makes it easy to deploy the proxy, as the link can just be sent via email to users. Again, we expect other tools to be used for recruiting participants and specifying tasks for them to do. The proxy is flexible enough that it can easily be integrated with such tools.

Step (2)*—Retrieving the Requested Document.* After the client's request has been received and analyzed, the proxy attempts to retrieve the document specified by the request. If no document has been requested (i.e., the *replace* parameter was absent) the proxy returns the default start page. Otherwise, the proxy opens an HTTP connection to the specified site and requests the document using either the GET or POST method, depending on which method the client used to request the page from the proxy. The proxy then downloads the document from the server. At this time, the rest of the data needed for the log entry is stored, including the HTTP response code and the final destination URL (in case the proxy was redirected by the server).

Step (3)*—Redirecting Links to the Proxy.* Before the downloaded page is sent back to the client, it must be edited so that all the links on the page are redirected through the proxy. This work is done by the *proxy editor* module. Initially, the proxy checks the content type of the page. If the content type is provided by the server and is not of the form `text/html`, the proxy assumes that the page is not an HTML document and returns it to the client without editing (e.g., a .pdf or .ps file). Otherwise, the proxy runs the page through the proxy editor.

The editor works by dynamically modifying all requested pages, so that future requests and actions will be made through the proxy. The document's base

HREF is updated and all links, including page hyperlinks, frames, and form actions, are redirected through the proxy.

First, the <BASE> tag is updated or added to the page within the page's enclosing <HEAD> tags. This tag's HREF field points to the document base—a location against which to resolve all relative links. This allows the client browser to then request stylesheets, images, and other embedded page items from the correct web location rather than through the proxy.

The proxy editor also modifies all link URLs in the page to use the proxy again on the next page request. Thus, once a person has started to use the proxy, all of the links thereafter will automatically be rewritten to continue using it. This includes standard text links as well as client-side image map links.

The editor also adds transaction IDs to each link. Again, transaction IDs represent the Nth page that a person has requested. Embedding the transaction ID into a link's URL lets the proxy identify exactly what page a person came from. Link IDs are also added to each link URL. This allows the proxy to identify exactly which link in the page a person clicked on. If the current page is a frame, the proper parent ID and frame ID parameters are also included.

The link URL and other parameters are included as variables within a query string for the link. The link URL itself is first rewritten as an absolute link and then added, along with the other parameters, to a query that will be read by the proxy. For example, if you are viewing the page www.yahoo.com through the proxy, the link

```
<A HREF="computers.html">
```

would be rewritten as

```
<A HREF="http : //tasmania.cs/webquilt/webproxy?
replace=http : //www.yahoo.com/computers.html&ti
d=1&linkid=12">
```

HTML <FRAME> tags are dealt with similarly—the URLs for the target frames are rewritten to pass through the proxy and extra information such as the frame parent's TID and the frame ID are included.

<FORM> tags are dealt with a little differently. The <FORM> tag's ACTION field is set to point back to the proxy as usual, but the actual target URL, the current TID, and, if necessary, the Parent ID and Frame ID, are encapsulated in <INPUT> tags with input type "hidden." These tags are inserted directly after the enclosing <FORM> tag. Since they are of type "hidden," they do not appear to the user while browsing, but are included in the resulting query string upon a FORM submit. The proxy editor also handles tags of the form <META HTTP-EQUIV="refresh" . . . >. These tags cause the browser to load a new URL after a specified time duration. The editor updates these tags to make sure the new URL is requested through the proxy.

The Proxy Editor implementation uses a simple lexical analysis approach to edit the page. The editor linearly scans through the HTML; comments and plain text are passed along to the client unchanged. When a tag is encountered, the type of the tag (e.g., 'A' or 'TABLE') is compared against a set of tags that

require editing. If the tag is not in this set (i.e., not a link) it is simply passed along to the client, otherwise it is handed off to the proper TagEditor module, which updates the tag contents as described above, before being sent along.

Steps (4) *and* (5)*—Page Caching and Logging.* After performing any necessary editing and sending the requested document to the client, the proxy then saves a cached copy of the HTML page. Before writing out to disk or to a database, the original document is run through the proxy editor again, but this time only the <BASE> tag is updated. None of the links are modified as in step 3. This feature was added to let web designers open up web pages locally and see the same content that test participants saw, making it robust to any changes on a web site. Finally, the log entry for the current transaction is written to the appropriate log file.

2.1.3 *Additional Proxy Functionality.* The base case of handling standard HTTP and HTML is straightforward. However, there are also some special cases that must be dealt with. For example, cookies are typically sent from web servers to client browsers. These cookies are sent back to the web server whenever a client browser makes a page request. The problem is that, for security and privacy reasons, web browsers only send cookies to certain web servers (ones in the same domain as the web server that created the cookie in the first place). To address this, the proxy logger manages all cookies for a user throughout a session. It keeps a table of cookies, mapping from users to domains. When a page request is made through the proxy, it simply looks up the user, sees if there are any cookies associated with the requested web server or page, and forwards these cookies along in its request to the web server. This is currently handled within the proxy by the HTTPClient library, with modifications made to ensure that separate cookie tables are used for each active user session. Once the user session is finished, the cookies for that session are cleared from the system.

Another special case that must be dealt with is the HTTPS protocol for secure communication. HTTPS uses SSL (Secure Socket Layer) to encrypt page requests and page data. The proxy logger handles HTTPS connections by using two separate secure connections. When a client connects to the proxy over a secure connection, the proxy in turn creates a new HTTPS connection to the destination server, ensuring that all network communication remains secure. Our implementation uses Sun's freely available JSSE (Java Secure Socket Extension) [Sun Microsystems 2001] in the underlying network layer to enable encrypted communication both to and from the proxy.

2.1.4 *Proxy Logger Limitations.* Trapping every possible user action on the web is a daunting task, and there are still limitations on what the WebQuilt proxy logger can capture. The most pressing of these cases is links or redirects created dynamically by JavaScript and other browser scripting languages. As a consequence, the JavaScript generated pop-up windows and DHTML menus popular on many web sites are not captured by the proxy. Other elusive cases include embedded page components such as Java applets and Flash animations. As technologies change and develop, the proxy will need to be updated to handle these new cases.

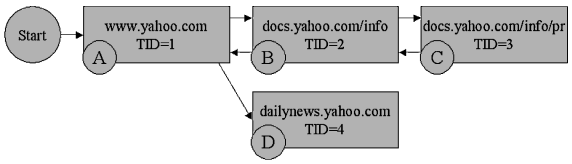

Fig. 7. A graphical version of a log file. The letters 'A', 'B', 'C', and 'D' are for this graph only and are not part of the log.

One obvious way to overcome these limitations is to use a traditional proxy approach, where all requests are transparently routed through a proxy. While this would certainly allow one to capture all user interactions, it introduces some serious deployment issues. Most significantly, the traditional proxy approach would require users to configure their browsers to use the proxy and then undo this setting after performing usability tests. This would seriously hamper the ease with which remote usability tests could be performed. Furthermore, any users who currently sit behind a firewall would be unable to participate, as changes to their proxy settings could render them unable to connect to the Internet.

Another situation that WebQuilt cannot handle is server-side image maps. With server-side image maps, the client browser sends the (x, y) location that the user clicked on, with the server returning the appropriate URL. Because the URL cannot be rewritten dynamically, we cannot yet change it. However, server-side image maps are no longer commonly found.

Finally, users may experience some additional delays with web sites when using the proxy due to overhead in retrieving and processing web pages. Whether or not these delays are noticeable and problematic still requires investigation.

2.2 Action Inferencer

Action inferencers transform a log of page requests into a log of inferred actions, where an action is currently defined as either requesting a page, going back by hitting the back button, or going forward by hitting the forward button. The reason the actions must be inferred is that the log generated by the proxy only captures page requests. The proxy cannot capture where a person uses the back or forward buttons of the browser to do navigation, since pages may be loaded from the local browser cache.

WebQuilt comes with a default *action inferencer*, but the architecture is designed such that developers can create and plug in new ones. It should be noted that given our logging approach, the inferencer can be certain of when pages were requested and can be certain of when the back button was used, but cannot be certain of back and forward combinations. Additionally, the current implementation does not specifically identify when a user clicks on the browser's refresh button.

As an example, Figure 7 shows a graph of a sample log file. Figure 8 shows how the default *action inferencer* interprets the actions in the log file. We know that this person had to have gone back to Transaction ID 1, but we do not know exactly how many times they hit the back and forward buttons. Figure 8 shows

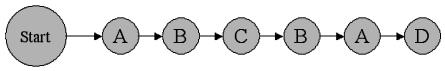

Fig. 8. One possible way of interpreting the log file in Table I. This one assumes that a person repeatedly hit the back.

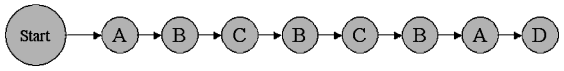

Fig. 9. Another way of interpreting the same log file. This one assumes that a person uses the back and forward buttons a few times before clicking on a new link (node D).

what happens if we assume that the person went directly back from TID 3 (node C) to TID 1 (node A), before going on to TID 4 (node D). Figure 9 shows another valid way of inferring what happened with the same log file. The person could have gone back and forth between TID 2 (node B) and 3 (node C) a few times before returning to TID 1 (node A).

2.3 Graph Merger

The *graph merger* takes all of the actions inferred by the *action inferencer* and merges them together. In other words, it merges multiple log files together, aggregating all of the actions that test participants did. A graph of web pages (nodes) and actions (edges) for the task is available once this step is completed.

2.4 Graph Layout

Once the log files have been aggregated, they are passed to the *graph layout* component, which prepares the data for visualization. The goal of this step is to give an (x, y) location to all of the web pages.

One algorithm that we tried was a simple force-directed layout of the graph. This algorithm tries to place connected pages a fixed distance apart, and spreading out unconnected web pages at a reasonable distance. The problem with this approach is that it is non-deterministic, with nodes being positioned at different places each time.

Currently, WebQuilt uses an edge-weighted depth-first traversal of the graph, displaying the most trafficked path along the top, and incrementally placing the less and less followed paths below. This algorithm also uses grid positioning to help organize and align the distances between the nodes. However, since there are a variety of graph layout algorithms available, we have simply defined a way for developers to plug-in new algorithms.

2.5 Visualization

The final part of the WebQuilt framework is the visualization component. There are many ways of visualizing the information. We have built one visualization that shows the web pages traversed and paths taken.

Web pages are represented by screenshots of that page as rendered in a web browser. These images are created by rerendering the saved HTML in

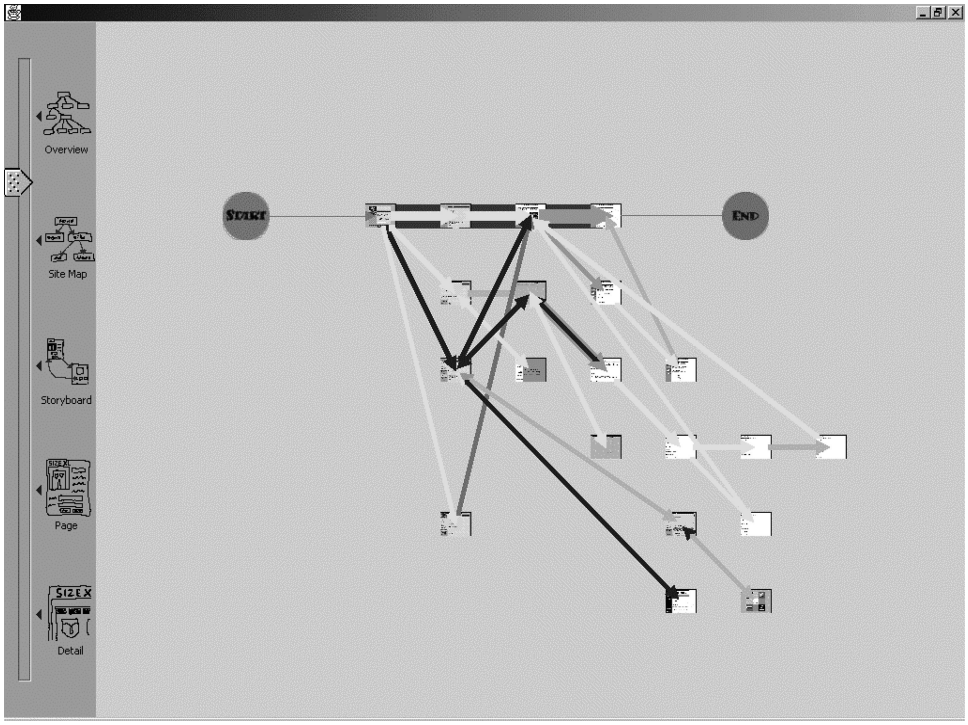

Fig. 10. An example visualization of twelve usage traces for a single defined task. The circle on the top-left shows the start of the task. The circle on the top-right shows the end of the task. Thicker arrows indicate more heavily traversed paths (i.e., more users). The thick black arrows along the top mark a designer indicated optimal path. Darker arrows indicate that users spent more time on a page before clicking a link, while the lighter shaded arrows indicate less time.

an internal browser (currently provided by NetClueSoft[2] and WindRiver[3]). By using the saved HTML, the visualization can display the pages as the user saw them. Arrows are used to indicate traversed links and where people hit the back button. Thicker arrows indicate more heavily traversed paths. A color scale is used to indicate the average amount of time spent before traversing a link, with lighter colors meaning short amounts of time and darker colors meaning longer amounts of time. Zooming is used to show the pages and paths at varying levels of detail. (See Figure 10).

Figure 10 shows an example visualization of twelve usage traces, where the task was to find a specific piece of information on the U.C. Berkeley web site. The pages along the highlighted path at the top represent the optimal, designer-defined path. By looking at the thickness of the lines, one can see that many people took the optimal path, but about the same number of people took a longer path to get to the same place. Following some of these longer paths, one can also see where users come to a page, and decide to backtrack, either via the back button or a link. Figure 11 shows a zoomed in view of one of the pages.

[2]NetclueSoftware, http://www.netcluesoft.com.
[3]WindRiver, IceStorm Browser, http://www.winriver.com/products/html/icebrowser_ds.html.

Fig. 11. The zoom slider on the left is used to change the zoom level. Individual pages can be selected and zoomed-in on to the actual page and URL people went to.

There are also several red arrows, which indicate that people took a long time before going to the next page. However, none of the red arrows are along the optimal path, meaning that people that took that path did not have to spend a large amount of time to get to the next page. One key feature of this visualization is the ability to zoom in and provide various levels of detail. For example, from the overview of the entire task, a viewer can see a red arrow indicating a long time spent on the page, but upon zooming in on that page the viewer would see that it is perhaps a very text-heavy page where the user probably spent time reading, or filling out a form. Providing the context of the task and a framework to add more details when needed, this visualization offers a number of simple, but very useful and quick analyses of the user experience (see Figures 12 and 13).

The graph in Figure 12 shows the results of 6 participants who were asked to find a specific product on the www.casadefruita.com web site. The thick arrows show that most participants had no problem finding the page with the information, however one user very quickly "ping-ponged" back and forth between the site map and various pages looking for the information. Figure 13 shows a trace of the same 6 participants at the same site performing two different tasks; finding out some information about the company, and purchasing an item. The two places in the graph with darkest arrows indicate the pages in the path participants spent the longest time in performing these tasks. The top zoomed-in image reveals that one of the pages is the order form. It makes

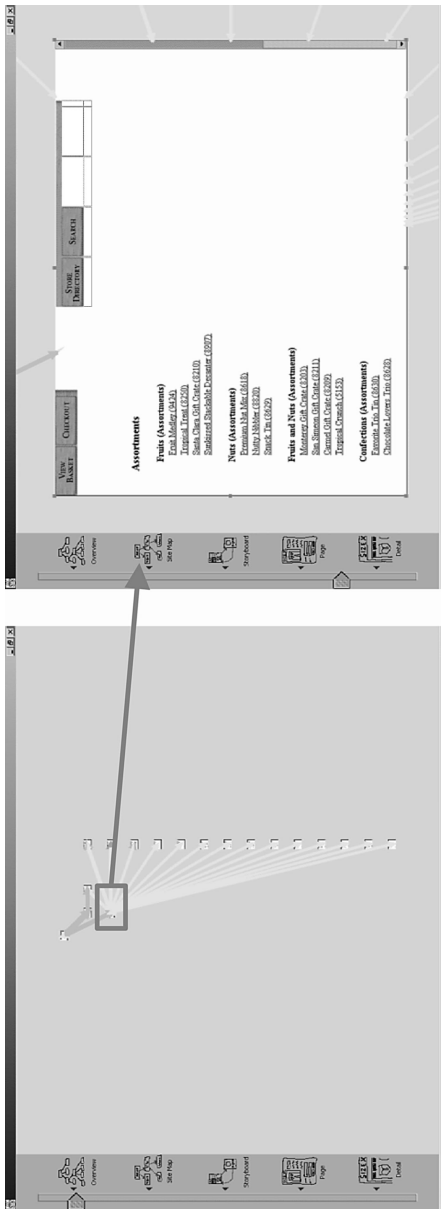

Fig. 12. In this example, six users were given the task of finding a piece of information about the Casa de Fruita company on their company web site (www.casadefruita.com). The thick arrows show that most users found this information in two clicks. However, one user quickly "ping-ponged" between the site-map page and many other pages before finding the information.

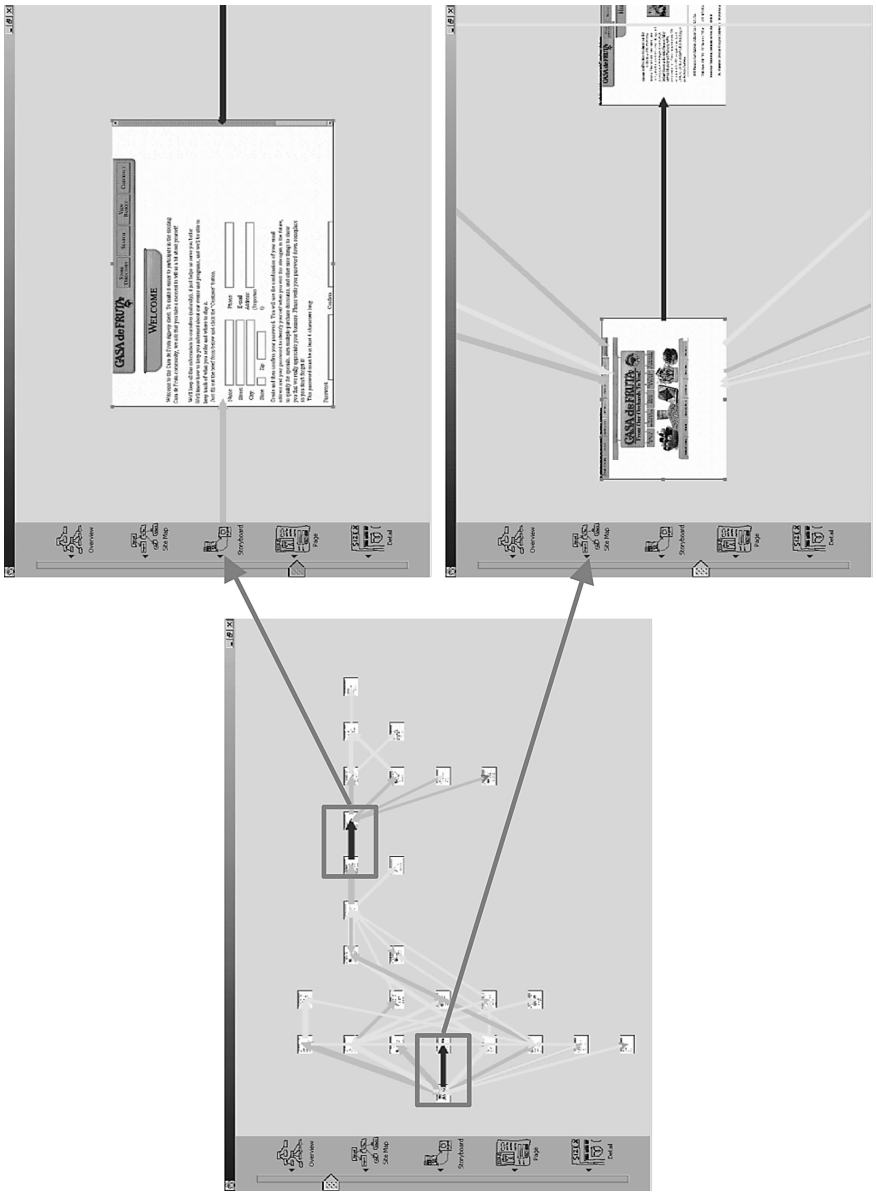

Fig. 13. In this task, six users were asked to find information on the Casa de Fruita web site, and to purchase a specific item. The darkest arrows show the places where users spent the most time, and zooming in, a designer can see that this would be an expected result. In the top instance, the user is filling out a form, and in the lower view, the user needed to read the text of the page to find the desired information.

sense that people would spend a lot of time here, as they have to enter shipping and credit card information. The lower zoomed-in image shows the company information page, which took many participants a surprising amount of time to find. It turns out that it was not clear that one of the images on the home page could be clicked on, confusing several of the participants.

3. RELATED WORK

In order to address some of the problems faced by client-side and server-side logging, the National Institute for Standards and Technology (NIST) has recently developed WebVIP [NIST 1999]. Intended as a tool to help run usability tests, WebVIP makes a local copy of an entire site and instruments each link with special identifiers and event handling code. This code is activated when a link is clicked on. WebVIP shares some of the same advantages that WebQuilt's logging software has, such as better compatibility with existing operating systems and browsers (since only the HTML is modified) and some ability to run logged usability tests on sites one does not own. However, WebQuilt's proxy approach to logging lets it work without having to download an entire site, which is more realistic for many situations. WebQuilt avoids the problems of stale content and of invalid path specifications for complex sites, and also works with database-backed sites that dynamically generate HTML when page requests are made.

Another system that is similar to WebQuilt's logging software is Etgen and Cantor's Web-Event Logging Technique (WET) [Etgen and Cantor 1999]. WET is an automated usability testing technique that works by modifying every page on the server. It can automatically and remotely track user interactions. WET takes advantage of the event handling capabilities built into the Netscape and Microsoft browsers. WET has more sophisticated event logging than WebQuilt currently supports, such as logging mouse motion, though there are plans for merging WET's advanced event handling capabilities into WebQuilt (see Future Work section). WebQuilt differs with its proxy approach, however, which again does not require ownership of the server and requires no changes to the server. These last two advantages are important when trying to accomplish web evaluations—the designers and usability team might not be *allowed* to make changes to or be given access to a production server.

Usability log visualization has a long history. Guzdial and colleagues review and introduce several desktop-based systems in Guzdial et al. [1994]. There are other recent visualizations that use the notion of paths. For example, the Footprints web history system [Wexelblat and Maes 1999] displays aggregate user paths as hints to what pages have been followed by other people. VISVIP [Cugini and Scholtz 1999] extends the work in WebVIP and shows individual paths overlaid on top of a 3-D visualization of the web site.

Two commercially available tools share similar goals to WebQuilt. Vividence Clickstreams [Vividence 2000b] visualizes individual and aggregate user paths through a web site. However, it uses client-side logging and has all of the problems associated with that technique. Furthermore, WebQuilt's visualization differs in that it combines aggregate path information with designated optimal

paths to make it easier to see which pages people had trouble with. WebQuilt also uses a zooming interface to show different portions of the web site, including screenshots of individual pages to provide better context.

Visual Insight's eBizinsights[4] is a sophisticated server log visualization tool for generating custom reports and interacting with the log data. While providing a number of interesting and useful chart-style visualizations, eBizinsights targets a marketing and management audience, which eventually will affect the designer decisions, but is not intended as a tool for the designer to use as he or she works. eBizinsights also uses server side logging, which, has all of the problems mentioned earlier with this technique.

The closest visualization work is QUIP [Helfrich and Landay 1999], a logging and visualization environment for Java applications. WebQuilt builds on QUIP by extending the logging and visualization to the web domain. WebQuilt also adds zooming, and uses screenshots of web pages for detail instead of abstract circles.

4. FUTURE WORK

We would like to extend our proxy logging system to overcome some of its current limitations, especially the handling of JavaScript. One possible approach would be to include a full JavaScript parser within the Proxy Editor module. The JavaScript code itself could then be updated to ensure that dynamically created links and pop-up windows pass through the proxy. Care must be taken, however, to ensure that the parsing and editing can be done in a reasonable amount of time, so the user's browsing experience is not adversely affected.

We would also like to capture a richer set of user interactions as they surf or perform a task. There is a wealth of data available to client-based loggers that our proxy currently does not collect. This includes page locations that are mistaken for links and clicked, page scrolling events, and more. By writing JavaScript code that captures these events and then inserting this code into pages as they are proxied, we may be able to remotely log some of these forms of interaction without any additional effort on the part of the user. We are currently investigating the approach used by WET [Etgen and Cantor 1999], which captures low-level events on the Mozilla browser and on Microsoft's Internet Explorer browser, as well as work done by Edmonds on tracking user actions [Edmonds 2001]. Using some of these technologies, however, could limit the variety of Internet-enabled devices which could use this proxy framework. Ideally, we would like WebQuilt to gather data from more than just traditional web browsers.

We are currently extending WebQuilt to capture and visualize the web experience on a variety of devices, such as PDAs and cellular phones. It is fairly straightforward to capture transactions for devices that have a standard HTML browser, but several modifications are needed to extend WebQuilt so that it can capture WAP-enabled phones.

We are also currently looking at additional visualizations for displaying and interacting with the traces. There has been some work done in visualizing server

[4]Visual Insights, eBizinsights, http://visualinsights.com.

logs [Edmonds 2001; Hochheiser and Schneiderman 1999; Pitkow and Bharat 1994] as well as visualizing individual and aggregate user paths [Wexelblat and Maes 1999; Cugini and Scholtz 1999; Vividence 2000b]. We plan to reimplement some of the ideas demonstrated by these visualizations, and add interactions and visualizations that are more useful for web designers. For example, the visualization in Figure 11 could be modified such that when zoomed in, the arrows could be re-anchored to show exactly which link was clicked on from a given page. This is an example of semantic zooming [Bederson and Hollan 1994] where the details of the visualization change depending on zoom level. Along the same lines, we would like to include methods for accessing more details, or filtering of information, depending upon the needs of the individual designer.

Finally, we plan on doing a rigorous evaluation of WebQuilt with real designers and usability experts to assess the overall quality of the tool, as well as the exploration of larger, more complex data sets.

5. CONCLUSIONS

We have described WebQuilt, an extensible framework for helping web designers capture, analyze, and visualize web usage where the task is known. WebQuilt's proxy-based approach to logging overcomes many of the problems encountered with server-side and client-side logging, in that it is fast and easy to deploy, can be used on any site, can be used with other usability tools such as online surveys, and is compatible with a wide range of operating systems and web browsers.

We have also described the architecture for WebQuilt, and shown how new algorithms and visualizations can be built using the framework. Again, we knew that we would not have all the solutions for analyzing and visualizing the captured data, so the system must be extensible enough so that new tools can be easily built. We have demonstrated one simple zooming interface for displaying the aggregated results of captured web traces, and are currently building more sophisticated visualizations and interactions for understanding the data.

The WebQuilt homepage can be found at:

`http://guir.berkeley.edu/projects/webquilt`

ACKNOWLEDGMENTS

We would like to thank James Lin and Francis Li for their feedback during the development of WebQuilt. We would also like to thank Kevin Fox, Tim Sohn, Tara Matthews, Andy Edmonds, and the Group for User Interface Research for their ideas and work on improving portions of WebQuilt. Finally, we would like to thank NetClue and WindRiver for providing us with copies of their Java web browser components.

REFERENCES

BEDERSON, B. AND HOLLAN, J. 1994. Pad++: A Zooming Graphical Interface for Exploring Alternative Interface Physics. In *ACM Symposium on User Interface Software and Technology*: *UIST '94.* Marina del Rey, CA, November. 17–26.

BEYER, H. AND HOLTZBLATT, K. 1998. *Contextual Design*: *Defining Customer-Centered Systems.* Morgan Kaufmann, San Francisco.

CHI, E., ET AL. 1998. Visualizing the Evolution of Web Ecologies. In *ACM CHI Conference on Human Factors in Computing Systems.* 400–407.

CHOO, C. W., DETLOR, B., AND TURNBULL, D. 1998. A Behavioral Model of Information Seeking on the Web—Preliminary Results of a Study of How Managers and IT Specialists Use the Web. In *Proceedings of the 61st Annual Meeting*, 35, 290–302.

CUGINI, J. AND SCHOLTZ, J. 1999. VISVIP: 3D Visualization of Paths through Web Sites. In *International Workshop on Web-Based Information Visualization* (*WebVis'99*). Florence, Italy. 259–263.

DAVISON, B. 1999. Web Traffic Logs: An Imperfect Resource for Evaluation. In *Ninth Annual Conference of the Internet Society* (*INET'99*). San Jose, CA, June.

EDMONDS, A. 2001. http://sourceforge.net/projects/lucidity/.

Ergosoft Laboratories. 2001. *ErgoBrowser.* http://www.ergolabs.com/ergoBrowser/ergoBrowser.htm.

ETGEN, M. AND CANTOR, J. 1999. What Does Getting WET (Web Event-Logging Tool) Mean for Web Usability? In *Proceedings of the Fifth Conference on Human Factors and the Web.* Gaithersburg, MD, June.

GOULD, J. AND LEWIS, C. 1985. Designing for Usability: Key Principles and What Designers Think. In *Communications of the ACM*, 28, 3, 300–311.

GUZDIAL, M., ET AL. 1994. *Analyzing and visualizing event log files*: *A computational science of usability.* Presented at *Human Computer Interaction Consortium Workshop.* Winter Park, CO, February.

HELFRICH, B. AND LANDAY, J. A. 1999. *QUIP*: *Quantitative User Interface Profiling*, http://home.earthlink.net/~bhelfrich/quip/.

HOCHHEISER, H. AND SHNEIDERMAN, B. 1999. *Understanding Patterns of User Visits to Web Sites*: *Interactive Starfield Visualizations of WWW Log Data.* Technical Report ncstrl.umcp/CS-TR-3989, University of Maryland. College Park, MD, February.

MYERS, B. A. AND ROSSON, M. B. 1992. Survey on User Interface Programming. In *Proceedings of the Conference on Human Factors in Computing Systems.* Monterey, CA, May. 195–202.

NetrakerCorporation. 2001. *NetRaker Suite.* http://netraker.com.

NIELSEN, J. AND LANDAUER, T. 1993. A mathematical model of the finding of usability problems. In *ACM INTERCHI'93.* Amsterdam, The Netherlands, April. 206–213.

NIST. 1999. *WebVIP.* http://zing.ncsl.nist.gov/webmet/vip/webvip-process.html.

NORMAN, D. A. 1990. *The Design of Everyday Things.* Doubleday, NY.

PITKOW, J. E. AND BHARAT, K. 1994. WebViz: A Tool for World-Wide Web Access Log Analysis. In *First International Conference on the World-Wide Web.* Geneva, Switzerland, May. 271–277.

PITKOW, J. E. AND PIROLLI, P. 1999. Mining Longest Repeated Subsequences to Predict World Wide Web Surfing. In *Second USENIX Symposium on Internet Technologies and Systems.* Boulder, CO, October.

RUBIN, J. 1994. *Handbook of Usability Testing*: *How to Plan*, *Design*, *and Conduct Effective Tests.* John Wiley & Sons, NY.

SPOOL, J. M., ET AL. 1998. *Web Site Usability*: *A Designer's Guide.* Academic Press, San Diego, CA.

SPOOL, J. AND SCHROEDER, W. 2001. Testing Web Sites: Five Users is Nowhere Near Enough. In *ACM CHI Conference on Human Factors in Computing Systems*, *Extended Abstracts.* Seattle, WA, April.

Sun Microsystems. 2001. *Java Secure Socket Extension.* http://java.sun.com/products/jsse/index.html.

TAUSCHER, L. M. 1999. *Evaluating History Mechanisms*: *An Empirical Study of Reuse Patterns in WWW Navigation.* MS Thesis, Department of Computer Science, University of Calgary, Alberta, Canada.

TEDESCHI, B. 1999. *Good Web Site Design Can Lead to Healthy Sales*, http://www.nytimes.com/library/tech/99/08/cyber/commerce/30commerce.html.

TSCHALAR, R. 2000. *HTTPClient V0.3-2.* http://www.innovation.ch/java/HTTPClient/index.htm.

VIRZI, R. A. 1992. Refining the Test Phase of Usability Evaluation: How Many Subjects Is Enough? *Human Factors, 34*, 4, 457–468.

VisualInsights. 2001. *eBizinsights*. http://www.visualinsights.com.
Vividence. 2000a. *Vividence Browser*. http://www.vividence.com/resources/public/solutions/demo/demo-print.htm.
Vividence. 2000b. *Vividence Clickstreams*. http://www.vividence.com/resources/public/solutions/demo/demo-print.htm.
WEXELBLAT, A. AND MAES, P. 1999. Footprints: History-Rich Tools for Information Foraging. In *ACM CHI Conference on Human Factors in Computing Systems*. Pittsburgh, PA, May, 270–277.
ZonaResearch. 1999. *Shop Until You Drop? A Glimpse into Internet Shopping Success*. http://www.zonaresearch.com/promotios/samples/zaps/Zap06.htm.
Zoomerang. 2001. http://www.zoomerang.com.

Received April 2001; revised May 2001 and June 2001; accepted July 2001