Click-once Hypertext: Now You See It, Now You Don't

Fred Douglis
AT&T Labs
douglis@research.att.com

Sonia Jain*
Princeton University
jain@phoenix.Princeton.EDU

John Klensin
AT&T Labs
klensin@research.att.com

Michael Rabinovich
AT&T Labs
misha@research.att.com

2nd Workshop on Internet Applications, July 2001

Abstract

SMASH is a system for automatically removing text from a web page as hyperlinks are selected. Making links (and optionally other related text) *click-once* allows a system to avoid clutter and provide visual feedback to a user. Unlike typical other adaptive hypertext systems, SMASH generates pages specially on the origin server but allows the browser to update the content automatically and efficiently. We discuss several trade-offs in the design of the system and a transparent and portable implementation using JavaScript.

1 Introduction

There are numerous examples of web pages that contain lists of many hyperlinks (also known as anchors). Some of these pages contain *transient* data such as news items that are typically viewed once via that page and then not again. Examples include search engines, news clipping services, and notification agents such as the AT&T Internet Difference Engine (AIDE) [4].

A web page can easily get cluttered with a large number of hyperlinks that have already been seen. Rather than simply relying on changes in text color or other techniques to differentiate them, one might avoid this clutter completely by removing those links along with the text associated with them. This can improve user efficiency by focusing on what is actually new and different for that user. Note that the choice of which information should be treated in this way must ultimately be with content authors: not all links will be "click-once."

Why isn't changing the color of the link when it is selected sufficient? First, changing the color does not reduce page clutter, and clutter has been shown to reduce user efficiency [11]. Second, there may be multiple page fragments associated with the link. All these fragments can be removed when the link is selected, while the color can only change on the the link text itself. Third, on handheld devices, screen real estate is at a premium, and colors are poor or nonexistent. Finally, a growing number of links are represented on the page by images rather than text, in which cases changing colors is impossible.

We refer to the process of hiding click-once links and their associated text as *elision*. Eliding a link need not be the same as removing it completely, because it may be important to provide a mechanism for occasionally restoring these links on demand. Restoration gives users the ability to back up and recover information that they had prior to clicking on the link, for instance to refresh their memory about where they found it.

Removal of click-once links can be done on the client or server. There is a wide body of work in the area of server-based adaptive hypertext [1], but a server-based approach for this problem would suffer from latency problems, if not scalability. Refreshing the page upon each

*This work was done while a summer intern at AT&T Labs.

followed link seems infeasible, because the server would need to know which links are followed, and then generate and transfer an updated version. The network latency to refresh from the server would make following a link appear sluggish (and perhaps painfully slow) in a wide-area environment. Furthermore, if the rate of followed links dominates the existing refresh rate of the page, it could increase the server load dramatically.[1]

With client-side support, the rewriting can be done much more efficiently. It shifts the burden to the browser and avoids additional network latency. However, the client-side approach presents some challenges. First, one needs a method to let the client know which links are special, and how to handle them when they are clicked on. Second, one needs to provide a means for the client browsing history to persist across page visits and browsing sessions. There are a number of options for how to support client-side elision; we have implemented one, which we describe in detail, and considered several alternatives as well.

Specifically, we have designed and tested a JavaScript interface that allows click-once links to be flagged in the enclosing HTML through a combination of an indirection through a JavaScript function in an anchor and an additional HTML tag. Click-once links are associated with text regions that are enclosed by a special HTML tag; these regions typically enclose the links themselves, and may enclose other text that is not part of the hyperlink, or multiple related adjacent hyperlinks. Note that the use of this elision technique is completely under the control of the page author. Elision may not always be appropriate; for example, a page with a hand-crafted layout might look incorrect if regions of the page were suddenly removed, and only the author (or authoring tool) knows which links may lend themselves to elision.

Our implementation uses cookies to record information about which links have been followed. The cookies ensure that the next time the user retrieves the page, the elided links will be omitted. Our use of cookies is rather unconventional. In addition to passing state to the server for optional use in subsequent page generation, our cookies store information for the JavaScript that executes on the client.

We refer to our elision system as SMASH, for the (admittedly awkward) acronym *Self-Modifying As Selected Hypertext*. The rest of this paper describes examples in greater detail (Section 2), considers some of the extra features that are useful in a system like this (Section 3), discusses implementation alternatives (Section 4), expands on an implementation of SMASH in the AIDE system (Section 5), discusses limitations and future work (Section 6), reviews related work (Section 7), and concludes.

2 Motivating Examples

Many web pages will not lend themselves to this sort of treatment, but there are a number of generic categories of pages that would appear to benefit. The greatest benefit will derive from pages that the server is already customizing based on what the user has seen in the past. The difference here would be that the customization becomes immediate rather than depending on a reload from the server and the delay that would entail.

2.1 Customized What's New Pages

Over the past few years, there has been a significant amount of work in notifying users of changes on the web. This typically takes the form of notification to users, via email or a web page, of a list of pages that have recently changed. AIDE [4], Mind-it [8], and the Informant [6] are examples of this service. It is also possible to use cookies to track which pages a user visits on a site, and customize that site's "What's New" page to reflect what the user has seen [3].

In any of these cases, a page that is presented to the user as being interesting (modified since it was last viewed) becomes *un*interesting as soon as the user sees the new version. When a service reports what is new, it removes those pages from the report the next time the report is generated. Removing the pages as the user selects them improves the user interface by making it clear what has and has not already been looked into. While this issue might seem trivial, considering personal experience, it is possible for these reports of new pages to have hundreds or

[1] Consider as an example the "My Yahoo!" page, `http://my.yahoo.com/`, which allows a user to specify a refresh interval. If a user updates the page every half hour, but follows two elidable news items in each interval, the page would be updated three times each half hour instead. Even a highly scalable service like that might prefer not to triple its load if alternatives are available.

thousands of items—if the user selects them haphazardly, rather than from top to bottom, he or she can quickly lose track of which ones have been processed.

To give a specific example (see Figure 1), AIDE reports the status of a single recently modified URL as a row of a table. There are several links per row, including one to open a new window with the differences between the current and most recently seen versions of the page described in that row, and another to open a window to modify parameters for tracking that page. One would want clicking the former to remove the page from the "What's New" list while clicking on the latter not to (since the page is still new relative to when the user last viewed it). Note that the link to show the differences is triggered by clicking an icon, rather than a textual anchor, and therefore the normal browser technique of coloring an anchor differently after it has been seen within a time interval does not apply. Instead, pages that have been viewed will continue to appear in the listing until it is refreshed by the server.

2.2 Portals

Another example of link clutter comes from the various portals provided on the Internet. Service providers such as Yahoo! and Excite@Home offer users a single page with customized items such as stock quotes, weather, news, and sports. One can specify a number of items to display in each category, and the service selects the highest priority items for each topic such as "world news" and "business."

Here, each article often has a unique URL associated with it. This means that once one has seen the article, changing the color provides a good indication that a "new" article is no longer new.

However, removing the link completely offers additional flexibility. For instance, one could remove it and replace it with a new link, which is provided in the HTML source but not displayed initially. Using an extension to HTML markup, the server would provide both the links and the framework necessary for the browser to display first one link and then the other. However, as with other types of client-side elision, the server would not need to be contacted at the time the new link is displayed.

This support would be most useful if the portal actually tracks what the user sees, so that pages that are updated periodically from the server do not simply reinstate a link that had been elided. This information can be passed back to the server via a cookie, as described in Section 4, or another mechanism such as `HTTP POST`. Furthermore, the same news item tends to be reported by portals in multiple places, such as separate "business" and "technology" categories or a per-company list of recent news items. Once the user has viewed a news item from anyplace it is presented, it might be removed from all the places where it would otherwise appear. (Again, while the server would have to provide mark-up that specifies when this is appropriate, the browser would do the "heavy lifting" of updating the various instances.)

2.3 Search Engines

Like portals and notification agents, search engines present a list of links to a user. Like portals, these links are well served by simply color-coding those that are followed, but they can benefit further by being completely elided after use. This would likely be a matter of personal preference, an option the user would maintain on the search page or via a cookie.

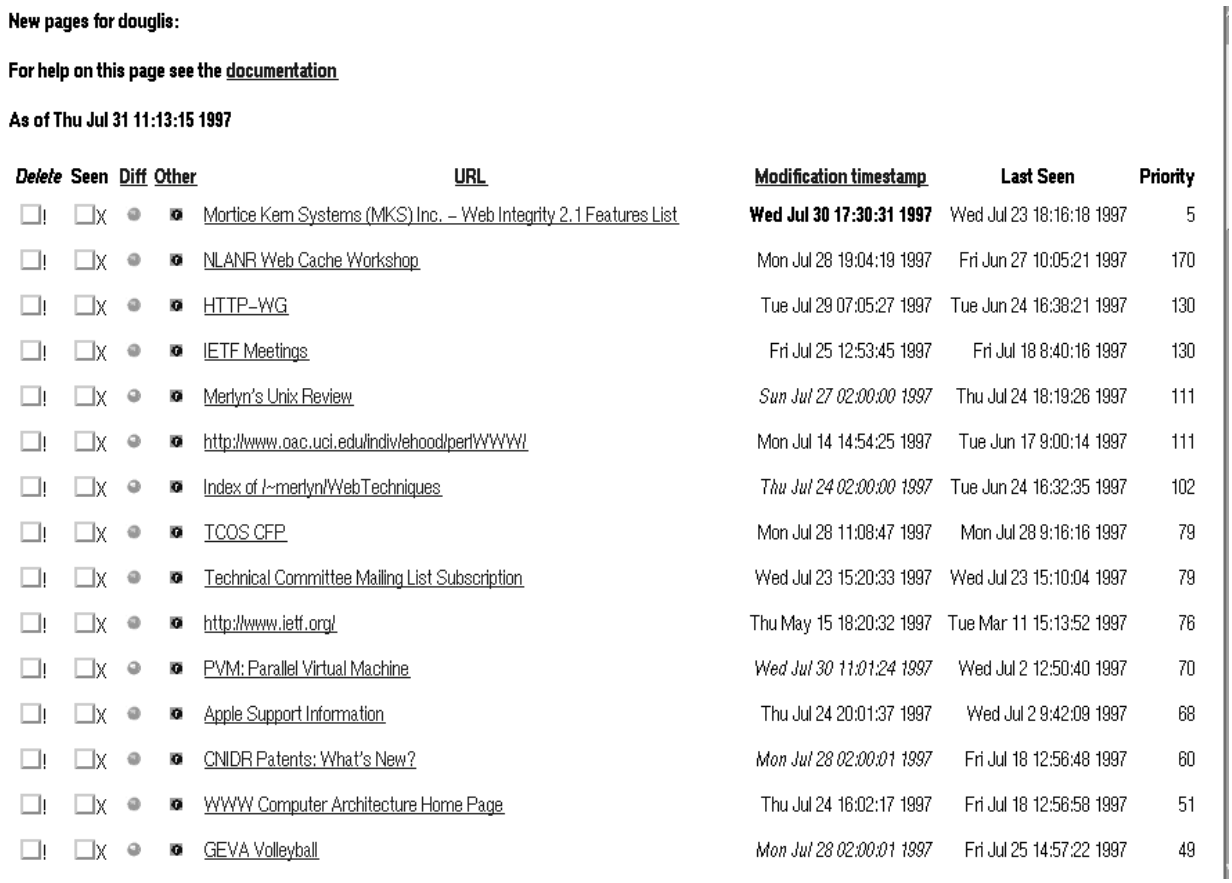
3 Features

Here we go into greater depth about what functionality is desirable in supporting click-once links. We have identified several features of interest, not all of which have been implemented so far:

Customization. Assuming that a server can support link elision, by tagging the HTML source appropriately, authors need to be able to specify whether links should be elided in the first place.

Immediacy. The effect of eliding a link should be visible immediately, not after a page reload. If one follows the link by opening a new window, still displaying the original page, the elision should instantly take place in the old window. If one follows the link in the same window, then the change should be apparent if and when the user backs up to the previous page.

Durability. In most cases, links that are selected should be elided not only in the user's current view of the enclosing page, but also in subsequent regenerated versions of that page.



New pages for douglis:

For help on this page see the documentation

As of Thu Jul 31 11:13:15 1997

Delete	Seen	Diff	Other	URL	Modification timestamp	Last Seen	Priority
☐!	☐X			Mortice Kern Systems (MKS) Inc. – Web Integrity 2.1 Features List	**Wed Jul 30 17:30:31 1997**	Wed Jul 23 18:16:18 1997	5
☐!	☐X			NLANR Web Cache Workshop	Mon Jul 28 19:04:19 1997	Fri Jun 27 10:05:21 1997	170
☐!	☐X			HTTP-WG	Tue Jul 29 07:05:27 1997	Tue Jun 24 16:38:21 1997	130
☐!	☐X			IETF Meetings	Fri Jul 25 12:53:45 1997	Fri Jul 18 8:40:16 1997	130
☐!	☐X			Merlyn's Unix Review	*Sun Jul 27 02:00:00 1997*	Thu Jul 24 18:19:26 1997	111
☐!	☐X			http://www.oac.uci.edu/indiv/ehood/perlWWW/	Mon Jul 14 14:54:25 1997	Tue Jun 17 9:00:14 1997	111
☐!	☐X			Index of /~merlyn/WebTechniques	*Thu Jul 24 02:00:00 1997*	Tue Jun 24 16:32:35 1997	102
☐!	☐X			TCOS CFP	Mon Jul 28 11:08:47 1997	Mon Jul 28 9:16:16 1997	79
☐!	☐X			Technical Committee Mailing List Subscription	Wed Jul 23 15:20:33 1997	Wed Jul 23 15:10:04 1997	79
☐!	☐X			http://www.ietf.org/	Thu May 15 18:20:32 1997	Tue Mar 11 15:13:52 1997	76
☐!	☐X			PVM: Parallel Virtual Machine	*Wed Jul 30 11:01:24 1997*	Wed Jul 2 12:50:40 1997	70
☐!	☐X			Apple Support Information	Thu Jul 24 20:01:37 1997	Wed Jul 2 9:42:09 1997	68
☐!	☐X			CNIDR Patents: What's New?	*Mon Jul 28 02:00:01 1997*	Fri Jul 18 12:56:48 1997	60
☐!	☐X			WWW Computer Architecture Home Page	Thu Jul 24 16:02:17 1997	Fri Jul 18 12:56:58 1997	51
☐!	☐X			GEVA Volleyball	*Mon Jul 28 02:00:01 1997*	Fri Jul 25 14:57:22 1997	49

Figure 1: AIDE's list of recently modified pages. (This figure originally appeared in [4].)

Reversal. A user should be able to view the page with all links included, or to back up and reinstate the previously elided link(s).

Transparency. Ideally, click-once links should not require special support from browsers, or explicit software installation or reconfiguration by users. Downloading a plug-in or ActiveX control once would not appear to be onerous. However, using functionality that users typically already enable, such as JavaScript, is preferable. Ultimately, if this idea were to gain sufficient popularity, support might be included directly in HTML and common browsers.

Performance. The impact on performance of supporting click-once links should be negligible. This implies that pages cannot be downloaded from the server on each link unless connectivity and server performance are such that regenerating the page is not discernibly slower than redisplaying it on the screen, something that will not be true except in limited cases. It also implies that the inclusion of click-once links should not affect the cachability of pages, or require that extra pages be downloaded.

4 Implementation Issues

We explored several alternatives for click-once links before settling on JavaScript. In the course of this explo-

ration, we came across a number of interesting implementation issues that bear discussion. Any implementation would require the following:

- Loading content.
- Pre-processing the content to elide previously clicked links prior to rendering.
- Intercepting clicks to elidable links, re-processing the page to elide the just-clicked link, and re-rendering the page.
- Storing the set of elided links in some way so that future visits to the page will render the page with those links elided.

4.1 Alternatives

Considering just widely available methods for active content, there are three basic alternatives to implement this functionality: a pure Java approach, a JavaScript/Java combination, and a pure JavaScript implementation. The pure Java approach would wrap the HTML page into a Java applet, passing the HTML content to this applet as a parameter. The applet would render the page itself in its area (which would coincide with the entire window) as well as catch and process clicks to elidable links. When the user leaves the page, the applet's last action before telling the browser to display the new page would be to upload to the server the current set of elided links, to be used by the server to pre-process the page when it was re-visited. This implementation would rely on the applet for rendering HTML pages. We preferred to leave rendering to the browser, as we thought the browser implementations are more likely to keep up-to-date with new HTML features than Java classes. For this reason, we did not pursue this approach further.

The JavaScript/Java combination relies on JavaScript for catching clicks, eliding links, and storing the elided link set, while it utilizes Java's LiveConnect to download the HTML page. Unfortunately, LiveConnect turned out not to be fully supported by Internet Explorer. Thus, we were unable to achieve our portability goal and abandoned this implementation even though it worked in Netscape Navigator.

In our next approach we attempted to utilize JavaScript. This was much more successful although there were some portability problems: Internet Explorer and Netscape do not behave the same when given the same piece of JavaScript (an example of this is discussed below in Section 5). However, by first determining which browser was being used, we are able to return Netscape and Internet Explorer specific code. The next subsection describes this solution in greater detail.

4.2 JavaScript Implementation

The JavaScript implementation uses a separate, cachable file called `smash.js`, which contains code to invoke when a link is clicked on and to undo previous elisions. This file is generic for all SMASH pages. Thus, once cached, it never needs to be downloaded. Excerpts of this file appear in Figure 2 (the straightforward functions `extractCookieValue` and `elideLinks` are not shown).

A SMASH page is wrapped in a page with three scripts as shown in Figure 3. The first script loads the SMASH file above. The second script has no executable statements and only defines a variable containing the page itself. Every elidable link on the page is identified by a numeric identifier ("5" in the example of Figure 3), which is passed to function `registerClick` when the link is clicked on using the `onClick` call. This function simply adds the link ID to the cookie associated with the page.

The third script is the one that actually gets executed on loading the wrapper page. It invokes a function that modifies the page by eliding all links contained in the current value of the cookie and displays the page. Thus, the first visit to a SMASH page involves the steps of loading the wrapper and invoking the `displayPage` function. The latter will find an empty elided link set in the cookie and display the original page. When a user clicks on an elidable link, `registerLink` adds this link to the elided link set in the cookie. Subsequent visits to the page will load the original wrapper and execute the `displayPage` function, which will bring the page to its current state by eliding all links in the clicked link set as specified in the current cookie of the page.

The above implementation displays clicked pages in the same window where the initial SMASH-enabled page was displayed. The elision becomes visible when the user

```
var cookieString = document.cookie;

function registerClick(linkID){
   var cookieValue = extractCookieValue(cookieString);
   cookieValue = cookieValue + linkID + " ";
   document.cookie = "ClickedLinks" + "=" + escape(cookieValue);
   cookieString = document.cookie;
}

function displayPage(page){
   var cookieValue = extractCookieValue(cookieString);
   currentPage = elideLinks(cookieValue,page);
   document.open();
   document.writeln(currentPage);
   document.close();
}
```

Figure 2: SMASH JavaScript functions, shared across pages.

```
<html>
<head>
<title> Skeleton of a SMASH page </title>
<script src = "SMASH.js"></script>
<script language = "JavaScript">
   var page = \
      "<body> \
      ...\
      <SMASH> \
         <A HREF='http://www.princeton.edu' onClick = 'registerClick(5)'>\
         </A>\
      </SMASH>\
      ...  \
      ";
</script>
</head>
<script>
   displayPage(page)
</script>
```

Figure 3: A skeleton of a SMASH page.

revisits the initial page, either by stepping back in the surfing history, entering the page URL explicitly, or clicking on a link to this page from another document. Figures 4 and 5 show an implementation that displays clicked pages in separate windows and makes the elision visible immediately in the parent window.

This implementation presents two interesting points. First, a more logical way of opening a successor in a new window would have been to use an anchor with the `TARGET="_BLANK"` attribute. However, we found that Netscape and Internet Explorer did not process this attribute in the same way. Thus, we pass the successor URL to the `registerClick` function in the HREF attribute of the anchor, and then open the successor page in a new window inside that function.

The other point has to do with making the elision immediately visible in the parent window. The challenge we faced was that browsers strip JavaScript code from a page when displaying it, storing the code in memory. Thus, a straightforward implementation where the function called on a click (i.e., `registerClick`) would modify and re-display the page does not work. Once the modified page is displayed, it contains no JavaScript code. In particular, the `page` variable is no longer defined. Thus, when the next elidable link is clicked on, the attempt to pass the `page` variable to the function specified in `onClick` attribute would cause an error. Instead, our `registerClick` implementation only adds the clicked URL to the cookie and reloads the wrapper page anew (the `history.go()` statement in Figure 4). When the wrapper is reloaded and before its page is displayed, the `displayPage` function will elide links according to the current cookie value, which now includes the just-clicked URL. The cookie will be passed back to the server, which can use the information to drive future page generations, or ignore it. Either way, it must be prepared to receive the cookie—a reasonable requirement given that the server is configured to generate SMASH tags in the first place.

Ideally, the above trick would not entail re-fetching it from the server since it would be found in the browser cache, but in practice this is a problem for dynamic data as discussed below in Section 5.1.

5 A Sample Application

To demonstrate the benefits of SMASH and to gain some practical experience, we modified the AT&T Internet Difference Engine (AIDE) [4] to optionally generate SMASH-compliant lists of new pages.

The front-end for AIDE is written in Perl, and consists of some scripts and library modules. The primary steps to make AIDE use SMASH were:

1. Take a static HTML listing and modify it to use SMASH, in order to determine the best way to annotate links. By doing this, we discovered that there is an incompatibility between the JavaScript implementations of Microsoft Internet Explorer (IE) 5.50 and Netscape Communicator 4.7: IE would expand character sequences consisting of a percent sign and two hexadecimal characters, while Netscape would not. Since AIDE itself encodes special characters in URLs with percent-escaped sequences, it is necessary for SMASH to encode something like `%20` (the space character) as `%2520` (the percent character followed by the code for the space character), when generating output for IE.

2. Create a module for SMASH, which adds a level of indirection to HTML that is printed. Instead of printing directly to standard output, the front-end invokes the print method of a SMASH object, which has been initialized to know whether to use SMASH. If so, it filters the HTML by escaping newlines, quotation marks, and percent signs. (In practice, the current implementation performs another function, which we discuss below in Section 5.1.)

3. Modify the generation of the HTML header and trailer. When SMASH is in use, the HTML header includes the JavaScript snippets above, loading the Javascript functions file and setting the rest of the file to a variable that is executed at load-time. The trailer provides the HTML snippet to allow the "undo" functionality, completes the variable assignment, and invokes the code to initialize the page.

4. Modify the routine that generates the list of new pages, so that it:

   - Initializes a SMASH object,

```
var cookieString = document.cookie;

function registerClick(linkID, clickedURL){
   var cookieValue = extractCookieValue(cookieString);
   cookieValue = cookieValue + linkID + " ";
   document.cookie = "ClickedLinks" + "=" + escape(cookieValue);
   cookieString = document.cookie;
   window.open(clickedURL);
   history.go(0);
}

function displayPage(page){
   var cookieValue = extractCookieValue(cookieString);
   currentPage = elideLinks(cookieValue,page);
   document.open();
   document.writeln(currentPage);
   document.close();
}
```

Figure 4: SMASH JavaScript functions supporting separate windows for successor pages.

```
<html>
<head>
<title> Skeleton of a SMASH page </title>
<script src = "SMASH-multiwindow.js"></script>
<script language = "JavaScript">
   var page = \
      "<body> \
      ...\
      <SMASH> \
         <A HREF='javascript:registerClick(5,\"http://www.princeton.edu\")'>\
         </A>\
      </SMASH>\
      ...  \
      ";
</script>
</head>
<script>
   displayPage(page)
</script>
```

Figure 5: A skeleton of a SMASH page displaying successor pages in separate windows.

- Uses this object for all printing,
- Optionally wraps each row containing a new URL with `SMASH` tags, and
- Generates the anchor to display a new page so that it optionally invokes `javascript:registerClick` to create a new window that loads the original anchor, an invocation of the AIDE CGI script to display the changes.

It took a (rusty) Perl programmer about a day to make these changes to AIDE, involving just a couple of hundred lines of new code and changes to several dozen print statements.

One aspect of the functionality that we have not yet implemented in AIDE is the use of the SMASH cookies to drive subsequent page generation on the server. The reason for this is simple: AIDE already knows when a link is followed, and the next time the list of new pages is generated, that link is already omitted.

5.1 Dynamic Pages

By moving from a static HTML file to a dynamically generated AIDE page, we confirmed what we feared was a problem with the current implementation of SMASH: the need to reload the page.

Our experimentation with the SMASH prototype had not resulted in an implementation that could dynamically modify the page on the client side without refreshing it via the `history.go(0)` JavaScript call. With a static file, this would result in an HTTP `GET If-Modified-Since` (IMS) call to the server, which would quickly respond that the file itself had not changed. A dynamic page results in the recreation of the page on the server, which for AIDE is computationally expensive and in any case defeats the purpose of using SMASH to elide links on-the-fly within the browser.

Attempts to convince an IE client that the page was cachable and to send an IMS request to the AIDE server, rather than an unconditional GET, were not successful without a nontransparent modification. This was apparently because of the presence of CGI features such as `?` in the URL. Instead, as an interim solution, we modified AIDE's front-end to write the HTML to a static file with a URL that has no CGI features and then redirect the client to load that file. As click-once links are followed, the browser checks to confirm that the page has not changed, and the HTTP 304 (not modified) response occurs immediately. This solution has the disadvantage that the redirection is apparent to the user. A better short-term solution is to allow the CGI front-end to be queried but to respond immediately with a 304 response rather than regenerating the page; however, this will require that the browser issue an IMS request even for a page with CGI features in its URL (`?` and `&`).[2]

6 Limitations and Future Work

Here we raise or review the major limitations of our current prototype, and discuss some directions for future work. We distinguish between implementation-specific limitations, which are problems that we have made at least some effort to address, and additional enhancements that we have not yet attempted.

6.1 Limitations

Handling dynamic content without page refreshes, as discussed in Section 5.1, is of foremost importance. It may be possible within JavaScript to update the current page without an explicit refresh, or it may be necessary to use a different technique such as explicit browser support or an intermediate proxy. (Using a proxy, the browser would still refresh the page, but it would be the proxy's responsibility to intercept the request and rewrite the page without contacting the server. We did not pursue the "personal proxy" approach [10] because of our emphasis on transparency, but we may try it to address some of the other problems.)

Similarly, the need to use a JavaScript variable to store the entire contents of the HTML page is inconvenient at best, as it requires that special characters be escaped, and it requires extra JavaScript processing before the page can be displayed. A proxy or direct browser support would avoid this problem, as would any enhancements to JavaScript to support this type of application.

[2] It appears that at least some browsers consider CGI features to take priority over an explicit expiration timestamp in the HTTP header, which is counterintuitive.

The problem with the IE implementation of JavaScript interpreting special characters (percent signs) differently continues to plague our prototype. It appears to depend on different versions of IE: a solution that worked one day on one computer (running IE 5.5) failed the next day on another (running IE 5.0).

Finally, opening a URL in a new window works if the JavaScript is tailored to *always* open it in the new window, using the JavaScript `window.open` call, but not using the HTML `_target` attribute or by telling the browser explicitly via the mouse.

6.2 Future Work

One highly desirable direction for future work would be evaluations, both quantitative and qualitative. Quantitative studies would measure the overhead of regenerating pages dynamically through JavaScript, and of the extra page revalidations that result in HTTP 304 responses. These issues are anecdotally of only moderate impact, but have not been formally evaluated.

User studies would help evaluate SMASH qualitatively: thus far, only one person has used SMASH extensively with a real application (a coauthor of this paper, using AIDE). A human factors study with a population of non-expert users would help to validate the benefits of the SMASH approach, or conversely, demonstrate any problems.

Another direction involves extending SMASH further. First, one of the suppositions of SMASH is that when a link is selected, it results in the successful loading of some other page. In practice, that page may be unavailable, or some other error might occur. It would be useful for SMASH to have feedback to know that a region can only be elided after a successful page load. Alternatively, and as a somewhat less useful solution to this problem, one could change the undo mechanism from being all-or-none to being incremental, so that a user could manually restore a region after an error without restoring all other regions with it.

Second, the existing prototype might be expanded to elide links in multiple windows within a single browser. Currently, links are numbered within a single document, and only that page is updated when a click-once link is selected. It should be possible to force an update to other open windows so that the same link is removed from each, but this will require that the cookies span documents and that each document is updated immediately when appropriate. Similarly, SMASH might also be expanded to allow multiple `SMASH` regions in a single page to be elided by a single click.

Third, as mentioned in the discussion of portals (Section 2.2), one might extend SMASH to replace elided links with other links that have lower priority and are not initially displayed. This could be done by extending HTML to permit a limit on some type of tag, for instance rows in a table, or by modifying the HTML on the client as with SMASH. Other modifications to the current page, besides eliding text, would also be possible.

Fourth, SMASH regions are identified implicitly by a counter. A better way to associate these regions with the links that elide them would be to name the regions, similar to internal anchors within an HTML page.

Lastly, SMASH might be better integrated into browsers, rather than relying on JavaScript support, which has the limitations discussed above as well as the possibility of being disabled entirely by a user. For instance, a browser could integrate the list of elided links with its existing history database, and elide selected links rather than displaying them in an alternate color.

7 Related Work

Adaptive hypermedia (AH) [1] is a field that bears many similarities to our click-once hypertext method. Understood broadly, AH refers to personalization of the client's display on the basis of past behavior. Users have different backgrounds and goals; AH systems can readily adapt to the needs of different users and display information and links most relevant to the user at hand [2]. AH is especially useful in educational hypermedia settings, where users may have different knowledge levels. In such cases, having the ability to personalize information to the user's knowledge level is imperative. Additionally, AH systems are useful when navigating through a large amount of hyperspace. They can provide support for users, by controlling which links are displayed and limiting the amount of hyperspace that can be traversed, thus preventing users from being lost in hyperspace [2]. In this sense, SMASH is a particular case of AH, as both schemes aim to assist the user in navigating through hyperspace efficiently and

expediently.

While both SMASH and AH work towards the same end, their means are rather different. AH schemes attempt to build user models based on specific characteristics like estimations of user knowledge, cognitive skills, and goals. The display and filtering of information is based almost entirely on the user model. Thus, these systems tend to draw extensively on machine learning and other artificial intelligence methods [5]. In contrast, our click-once method adjusts a user's display based purely on the links the user has visited and does not involve a user model. Consequently, our scheme is able to perform all of the work on the client side, while traditional adaptive hypermedia applications tend to work on the server side as they require extensive state information about users and significant processing.

There have also been proposals to personalize Web pages based on various user characteristics such as location, access history, time of access, etc. (e.g., Kiyomitsu, et al. [7]). While not using machine learning, these proposals still require extensive server-side information to generate pages and are therefore inherently server-based. SMASH link elision, on the other hand, is driven by simple syntactic rules and is performed on the client.

Finally, the use of client-side scripting to go beyond native support for HTML is by now, of course, well-established. E-commerce sites commonly use JavaScript because most users enable it in their browser, and because it is reasonably portable. Examples of these include scripts that check user inputs before passing them to a web server, and links that open multiple windows on a single click. One recent research system that uses JavaScript when a page is actually loaded is *Timekeeper* [9], which causes a user's browser to record page download latencies.

8 Conclusions

We have described a portable and transparent system for supporting click-once hyperlinks with minimal interaction between the browser and the content provider. As a proof of concept, we believe it demonstrates the desirability and feasibility of our approach. Nevertheless, it has some shortcomings that could be addressed through future research and by better integration with web servers and browsers. These shortcomings and improvements were summarized in Section 6.

Ultimately, if SMASH is adopted more widely, one would want servers and browsers to support it without the need for convoluted JavaScript code, page refreshes, cookies, and so on. Simply tagging a region with SMASH and a link as click-once should be sufficient for a browser to elide the appropriate area. Using cookies as a method for accumulating which links have been followed and propagating that information back to the server would also prove useful for later customizations, but would naturally also have interesting privacy implications as well.

Acknowledgments

Robin Chen, Bin Wei, and anonymous referees provided helpful comments on earlier drafts of this paper. We are grateful for their assistance.

Excite@Home, Internet Explorer, Java, JavaScript, Netscape Communicator, URL-Minder, Windows, Yahoo!, and all other brand or product names are trademarks or registered trademarks of their respective holders.

References

[1] P. Brusilovsky, A. Kobsa, and J. Vassileva, editors. *Adaptive Hypertext and Hypermedia*. Kluwer Academic Publishers, Dordrecht, 1998.

[2] P. Brusilovsky and L. Pesin. Visual annotation of links in adaptive hypermedia. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 2 of *Short Papers: Agents and Anthropomorphism*, pages 222–223, 1995.

[3] F. Douglis. Server-side tracking of new documents. In *Proceedings of the 1st International Workshop on Web Site Evolution*, Atlanta, GA, Oct. 1999.

[4] F. Douglis, T. Ball, Y.-F. Chen, and E. Koutsofios. The AT&T Internet Difference Engine: Tracking and viewing changes on the web. *World Wide Web*, pages 27–44, Jan. 1998.

[5] K. Höök, Åsa Rudström, and A. Waern. Edited adaptive hypermedia: Combining human and machine intelligence to achieve filtered information. In *Workshop on Flexible Hypertext*, Apr. 1997.

[6] Informant. `http://informant.dartmouth.edu/`, Dec. 1996.

[7] H. Kiyomitsu, A. Takeuchi, and K. Tanaka. Web reconfiguration by spatio-temporal page personalization rules. In *Symp. on Applications and the Internet (SAINT)*, pages 75–84, Jan. 2001.

[8] Url-minder. `http://www.net-mind.com/URL-minder/URL-minder.html`, Dec. 1996.

[9] R. Rajamony and M. Elnozahy. Measuring client-perceived response times on the WWW. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS'01)*, pages 185–196. USENIX, Mar. 2001.

[10] H. Rao, Y. Chen, M. Chen, and J. Chang. A proxy-based personal portal. In *Proceedings of the WebNet99 Conference*, Oct. 1999.

[11] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Publishing, third edition, 1997.