Reducing Cognitive Overhead on the World Wide Web

Rebecca J Witt **Susan P Tyerman**

School of Computer and Information Science
University of South Australia

{Rebecca.Witt, Sue.Tyerman}@unisa.edu.au

Abstract

HyperScout, a Web application, is an intermediary between a server and a client. It intercepts a page to the client, gathers information on each link, and annotates each link with the discovered information. This paper reports on the development of *HyperScout var UniSA*, a development of the HyperScout model and application, that dramatically extends static and dynamic link annotations. Annotations provide the user with additional information, which they use to make better navigational choices. On the web, it is common for long lists of hyperlinks to be presented to the user, from which they select links to follow or ignore. The user's mental state in this situation is termed *cognitive overhead*, a potentially overwhelming condition. To assist the user in making their choice, various characteristics of a link may be presented to the user. Despite these characteristics being readily available from a number of sources, current web servers and browsers do not attempt to retrieve, let alone display, such attributes. To show that cognitive overhead is easily, and immediately, reducible, a number of techniques were explored. Development progressed from statically created annotations, through to dynamically generated annotations. The static annotations were implemented with a combination of tools available to every web author. It was found that, while simple enough for every author to implement, static annotations bearing static information would not be accurate or timely enough to guide the user. Therefore, information must be gathered dynamically. The solution is either an intermediary between server and client, or a more sophisticated browser.

Keywords: hypertext, navigation, world wide web, cognitive overhead.

1 Introduction

A web user's navigation trail is a complex mix of forward linear traversals, cyclic paths, leaps, dead-ends, and backtracks. To increase the efficiency of their navigation, a user visits a search engine and enters words that have relevancy to the topic being investigated. The search engine presents a list of, supposedly, relevant sites. The user is now faced with many decisions: which links are the most relevant, completely irrelevant, authorities on the subject, fast, broken, up-to-date, in French, *etc.* Instead of being a relatively transparent process, the search mechanics impinge strongly on the user's consciousness, leading to *cognitive overhead*.

Conklin (Conklin 1987) defines cognitive overhead as the burden of *meta-level decision making*, i.e. of making decisions about decisions. Subconsciously, a user may decide that the relevance, authority, speed, and language are the criteria against which a web page is measured. Essentially, cognitive overhead is the problem of deciding upon which criteria a link choice will be based.

This paper proposes that *cognitive overhead* on the World Wide Web can be decreased immediately. It explores a range of techniques with which to do so, both from the perspective of the author and web client.

Some background to the work is reported, followed by an explanation of the important issues. Section three and four describe the model and implementation of the system, *Hyperscout var UniSA (HvU)*. The remaining sections describe the evaluation of the system and ends with some concluding remarks.

2 From Hypertext to the World Wide Web

The World Wide Web (WWW) is comparable with a hypermedia system, hypermedia being a generalisation of hypertext. As such, the WWW inherits the disadvantages of hypertext, identified by Conklin as the impairment of user navigation due to disorientation and cognitive overhead. In fact, these problems are heightened on the WWW, as association, that is the specification of page relationships, and physical characteristics are uncontrollable. Hypertext-specific remedies are applicable to the WWW, but WWW-specific solutions are also required.

2.1 Hypertext

Human thought is not linear, but moves instantly to an associated thought. Hypertext attempts to model this process with hyperlinks. To be truly effective, hyperlinks require a minimal user effort and low cognitive overhead. Conklin also recommends that the time delay when traversing a link be as small as possible. Neglecting this adds to user disorientation. While the time delay may be controllable in hypertext systems, hypermedia systems add another level of unpredictability.

While nodes in hypertext are of similar size, node sizes in hypermedia vary greatly. From text of a few tens of kilobytes, to graphics of a few hundred kilobytes, to video of tens of megabytes, link transition times necessarily vary. This situation is echoed in the WWW, and is compounded by network delays.

Loeb (Loeb 1992) introduces the concept of *network advice*, notifying the user of potential delays in the

system. If uses can anticipate delays, they will not lose patience when attempting to retrieve information. It is often the unexpected delays that increase disorientation. Network advice is one solution for reducing problems associated with cognitive overhead.

2.2 Hyperlinks and Association

Linking is analogous with the association process of the human mind. However, "Links, in general, tie together concepts that have a natural association *in the mind of the person creating the links*" (emphasis added, Schnase *et al.* 1993). This is reflected in the WWW, where users find it difficult to understand an association an author meant to convey via a link.

Various models have been proposed to clarify associations (or the lack thereof). Akscyn *et al* identify a number of design issues in hypermedia (Akscyn *et al* 1987), and Landow has a set of rules for authors that are still relevant today (Landow 1991). These are attempts to force authors to adequately structure their hyperdocuments. Unfortunately, coercing web authors to follow conventions is an impossible task, as it is unenforceable.

2.3 Disadvantages of Hypertext

Conklin identifies two general problems of Hypertext that are pertinent to this research. Both affect a user's navigation if steps are not taken to avoid the problems of disorientation and cognitive overhead.

2.3.1 Disorientation

A user becomes disorientated when they lose track of where they are, or when pages are complex, contain unexpected content, or include internal and broken links.

Clicking an internal link moves the focus to a different location in the same page. This action is not indicated to the user, and the only means of judging their new location in the document is by the vertical scroll bar. The user may not even be aware that they are within the same document. Not knowing the association between the old and new location worsens the situation.

Broken links are particularly disorientating, especially in the current versions of browsers. A browser receiving a numeric error message replaces the current page with a new page displaying the error. The user is required to interpret the message and to back-button to the original page. This seems inefficient, potentially confusing the user. The browser could simply display a message box without displacing the original page, thereby allowing the user to retain a sense of context.

2.3.2 Cognitive Overhead

As previously defined, cognitive overhead is decision making about decisions. Currently, most of the decision criteria available for a link are hidden from a user. Attributes, such as author, modification date, and server status, are available from a number of sources, but the only attribute browsers present to the user is the URL. Only after traversing a link may a user discover this extra information. It is then too late to decide the link is not worth following.

2.4 Recent Work

Three recent studies that attempt to resolve some of the issues addressed previously are presented here. All annotate links with additional information, with varying degrees of success. These applications all work as plug-ins to existing browsers.

Link Lens (Stanyer and Procter 1999) provides an abstraction mechanism for the link, with two elements, *channel* and *site*. *Channel* represents a connection between the client and the server. *Site* represents the documents on the server. For these elements, a *Quality of Service summary* is calculated. Stanyer and Procter divide the user's link evaluation process into two parts:

- an assessment of content and
- an assessment of download *Quality of Service*.

The content component of Link Lens is created from META tags of target pages and information derived from the link's URL. Link Lens provides attributes such as *author* and *title*, and analyses a page for the distribution of keywords, a thumbnail outline of the document, and the number and size of embedded media and links. The major disadvantage of this approach is the intrusiveness of the display as a substantial portion of the window is covered by the thumbnail and other displayed data.

Visual Preview (Kopetzky and Mühläuser 1999) generates a visual description of a target page. This thumbnail is displayed when a link is moused-over. Their reasoning for this technique is based on user recall. Users will associate images with particular pages, and in this way, links can appear familiar to them. The disadvantage of Visual Preview is the increase in network traffic. Each target link is downloaded so that thumbnail images can be generated. In contrast, Link Lens only requires the header response for each page as the header contains the META tags. However, Visual Preview does have an advantage over Link Lens and HyperScout. META tags are author generated information, so Link Lens and HyperScout are at their most effective when the author has provided information. When the author does not provide META tags, the information provided by Link Lens and HyperScout is adversely affected. Visual Preview, on the other hand, delivers the same level of quality irrespective of the amount of information provided by the author.

An additional problem with thumbnails is the growing prevalence of corporate and standard formats for Web pages which effectively reduce the recognition factor for the user once the small image is rendered.

HyperScout (Weinreich and Lamersdorf 2000) is similar in concept to Link Lens. META tags are gathered and elements extracted from URLs, and results presented in pop-ups that appear next to each link when the link is moused-over. HyperScout appears to be superior to the previous two systems in two areas:

- Weinreich and Lamersdorf present an analysis of different techniques for displaying additional link information. From this comes their justification for the choice of implementation - the pop-up. The authors of the previous prototypes did not include justification.
- The system is more informative than Visual Preview, and less obtrusive and more attractive than Link Lens.

However, the HyperScout system is not perfect being in an experimental, rather than commercial, prototype stage. HyperScout has an inconsistent presentation. Currently, whatever information is available is displayed. One link may have only a few attributes, and for another link the information may be quite detailed. This inconsistency is likely to confuse novice users.

3 Model

This section describes the extensions to the HyperScout model that have been realised in the *HvU* system.

3.1 Users

Novice WWW users are initially impressed by the abundance of resources, though they are soon frustrated when attempting to retrieve information from these resources (Weinreich and Lamersdorf 2000, Kopetzky 1999). All users face the same difficulty, but experienced users have an understanding of which criteria to use in making a link choice. *HvU* makes explicit the information available for each link, perhaps bringing awareness to the user of the possible criteria sooner than if the information remained hidden.

Pay-by-the-minute or pay-by-the-byte ISP customers are interested in the fastest links and the smallest downloads. In contrast, experienced users (and no-restrictions ISP customers) may ignore fast links in favour of the most relevant link. Relevancy is gauged on a number of attributes including *keywords*, *author*, and the URL's domain, all of which the model presents to the user. Additionally, all users wish to avoid wasting their time with documents that cannot be accessed, or cannot be interpreted once downloaded.

3.2 Data Model

This section details the attributes created for each link (the output) and the sources used to generate the attribute values (the input).

Hyperscout categorises link attributes into five classes. *Content* attributes are *title*, *author*, *keywords*, *description*, *language*, *last modified date*, and the *first few lines* of a page. *Content* attributes are the resource for determining a link's relevance.

Access attributes are the *status* of a page (based on HTTP codes) and the file *size*. The HyperScout model displays the file size when it is greater than 30 Kbytes. *HvU* indicates the approximate magnitude, i.e. small, medium, or large, to aid the comparison of link sizes. The size thresholds may be set by the user, for example, greater than one Mbyte is medium, greater than 5 Mbyte is large. A future model would calculate the magnitude proportional to the current connection speed, i.e. the slower the connection the lower the thresholds.

The *Usage* attribute, of the HyperScout model, is the time of the user's *last visit* to the link. If the user is returning to a link in the same day, the message is displayed as "Last visit *x* minutes ago," otherwise the date and time are displayed. A future model would alter a link anchor's colour according to the time since the last visit. Current browsers indicated *previously visited* with a distinct colour that eventually times-out, i.e. is set back to *not visited*. Instead, the anchor colour could fade or change colour over time (Chen 1999). Even this, though, is ineffective in pages that do not use the 'standard' colours (blue for unvisited, purple for visited). In addition the model could display the length of time the user spent at a particular page, perhaps reminding the user of a page they consider important. Moreover, if a user clicked on a link and back buttoned immediately, perhaps even before any portion of the page is displayed, the anchor colour would not change to *visited*.

Topological attributes, of the HyperScout model, indicate the spatial relation between the current page and the pages to which it links. A link is labelled *Reference* if it jumps to a different location within the same page. This use of the term *Reference* is a departure from the traditional academic semantics of the term and is more like the associative link described by Conklin. A link to a different host is labelled *External* and, if there is no error, the delay time is displayed. Otherwise "Server is probably unreachable!" is displayed. *Cluster*, *Survey*, *Detail*, and *Associative* indicate the relationship between pages on the same server.

- *Cluster* links to a page in the same directory.
- *Survey* means the link is to a page in an ancestor directory of the current page.
- *Detail* links to a page in a descendant directory.
- *Associative* represents links to pages existing in non-overlapping directories.

Links are labelled as *Query* where the URL features the ? symbol or as *Home* where the links are to the homepage of the domain. In addition to these original HyperScout attributes, *HvU* implements attributes to represent the global topology of links. The domain's *Country* is listed explicitly, as is the *Domain* type. While the original attributes help reduce users' disorientation, the new attributes educate users on the components of URLs. *Country* and *Domain* are also useful in judging relevance. For example, if a user is searching for information on the system of government in Iceland, a URL containing *gov.is* may be missed unless Iceland and Government are displayed.

The *Action* attribute, of the HyperScout model, indicates whether a link opens a new window, removes a frame, controls another frame, or performs some JavaScript action. This attribute is very beneficial in avoiding user disorientation - the user is warned before an unexpected action occurs.

The *Format* attribute, of the HyperScout model, describes the file type, for example, *Audio* or *XML* file. A possible extension to this is to inform the user as to which applications or plug-ins can view a type. Better yet, it would indicate if the user has the applicable software installed, unlike the present situation where a user must traverse a link only to discover they may not be able to view the page.

The above link attributes are derived from four sources: the URL of a link, the other HTML attributes of the link's anchor tag, the META tags in the target page, and HTTP response codes. All of these sources are immediately and easily accessible.

The *Content* attributes are extracted from META tags. *Access* attributes are derived from the HTTP response header. The *Format* attribute is derived from the MIME type of a link, which is included in the HTTP response header.

The *Topological* attributes are all derived from a link's URL. *Reference* is the substring following a # symbol; *Query*, the substring following a ? symbol. *External* is derived from a comparison of the current host and the link host (if they are different, then the link is external). *Cluster etc.* is calculated by comparing the lengths of the URL paths, where one path is the prefix for the other path. If the link's path is shorter, it is a link to a higher directory (*Survey*). If the link's path is longer, it is in a lower directory (*Detail*). When neither URL path is a prefix for the other, then the link is *Associative*. If the URL path is null, i.e. the URL is just the host, then the link points to the *Home* of the server.

Action attributes are derived from other HTML attributes in the link's anchor tag. If the HTML attribute `target` has a value `_blank`, then a new window opens. If `target` is `_parent`, a frame is effectively removed (i.e. the target page is loaded into the whole window, not just the frame). If `target` is any string not beginning with an underscore _, then the link controls another frame.

3.3 Functional Model

HvU does not alter the existing functional model. This section describes the existing functional model of HyperScout.

While a page is downloaded to a user's browser, the page is parsed for HTML elements containing a hypertext reference (`href`), i.e. `a`, `area`, `link`, and `base` tags. For each link target, data is collected from the four sources, if available. The resulting information is dynamically inserted into the page, thereby annotating the respective link. The manner of annotation is discussed in Section 3.4.

When and what gathers the data and inserts the annotations is an important question. Constantly modifying existing pages is obviously impractical, so the creation of annotations must occur sometime between the server sending a page to the browser displaying the page. There exist three possibilities for what inserts annotations: the server, a client side proxy, or the browser.

HyperScout is a client side proxy that performs three tasks: records the user's browsing history, generates *some* additional link information, and inserts the annotations. To provide all data available, it would be necessary for the client to pre-fetch HTTP headers for all link targets. Given concerns over increased network congestion, the HyperScout client only collects data available from a link's URL, or from the HyperScout database (for each page visited, all the attributes of the page are recorded in the database).

To provide all available data, HyperScout may also be run as a server. HyperScout-as-server appends additional data to all anchor tags of an outgoing page. The additional data is extracted from the server side database that contains meta data for each page on the server as well as data on every page referenced by a server page. Section 5.4 provides a summary of the architecture of HyperScout.

However, HyperScout does not take enough advantage of a user's bookmark file. While HyperScout-as-client parses the bookmark file and adds the URLs to the database, there is no indication that the corresponding attributes for each page and each referred page are updated.

The future version of *HvU* will have the following functionality in regard to the bookmark file. It is essentially a supplement to the functionality provided by Internet Explorer 5 (IE5). IE5 allows a user to *subscribe* to a page, whereby IE5 periodically checks such a subscribed page for updates. Where IE5 checks if a file's modification date has altered, this model essentially determines if a page's *context* has altered. IE5 cannot give an indication of what has altered on a page (comparing only modification dates), but with the information provided by *HvU*, the user may have a better chance of detecting *what* has altered (such as text or graphics).

Also, *HvU* would examine every link on frequently visited bookmarked pages. This is justified in regard to network congestion. The links within a frequently visited site are also likely to be frequently visited. If the referenced pages have not altered, then network congestion has been increased needlessly. If *HvU* increases network traffic with extra header requests, the long term effect would be a decrease in congestion as the user need not request entire pages, their associated images or other data related to the pages.

3.4 Graphical User Interface Model

Weinreich and Lamersdorf include a brief survey of the different techniques for displaying link annotations (Weinreich and Lamersdorf 2000). These include: an *overview map*, a *reserved area*, *insertion after links*, *link colours*, *mouse pointers changes*, and *pop-ups*. Pop-ups were chosen for the HyperScout model (see Weinreich and Lamersdorf 2000 for justification).

A pop-up appears as a yellow rectangle when a link is moused-over. The pop-up appears near the link, but its exact location varies according to the positioning of the link in relation to the browser window. For example, if the area below the link is smaller than the size of the pop-up, the pop-up will appear above the link.

Consistent with HyperScout look-and-feel, *HvU* includes icons for each new attribute. The new icons and their interpretation are: $ symbol for *Company*, the earth for *Country*, a mortar board for *Education*, a crown for *Government*, a tank for *Military*, a net for *Network*, and the UN and Red Cross logos for *Organisation*. The *HvU* model allows the user to turn off either the icons or the attribute name, to reduce the size of the pop-up.

HvU allows the user to specify which attributes to display. This essentially implements that aspect of consistency addressed by Weinreich and Lamersdorf as future work.

There are two basic methods for interacting with a hyperlink anchor, thereby triggering the appearance and disappearance of a pop-up: the mouse click and the mouse-over. Each brings its own advantages and disadvantages.

The mouse click requires at least two actions from the user: the mouse-over (pointing the mouse at the desired location) and a mouse button click. Popular browsers (Netscape and IE) use this combination to activate the link, so a different combination is required for the pop-up to activate, i.e. right mouse button or keys, is required. This, too, has its own difficulties *vis-a-vis* standard Apple computer mice which have only one button. The advantage this technique has over the mouse-over technique (below) is the pop-up information widget remains active until the user clicks outside the widget.

The mouse-over (or hover) requires the least effort, but perhaps more dexterity, from the user. The user moves the mouse over the relevant hyperlink anchor and the information window pops up. However, this technique has the potential to become a visual intrusion if the user moves the mouse randomly about the page. This shortcoming is avoided by introducing a delay. If the user hovers over a link for at least one second, then the system assumes the user has intentionally moused-over the link.

This technique has been implemented for multiple-choice quizzes in an online tutorial to coach students on the Java programming language at the University of South Australia. In this particular implementation students can either make a choice immediately or hover over their choice to see the hint within the pop-up. The system records the number of correct and incorrect choices students made as well as the number of hints the student received.

The main disadvantage of the hovering technique, compared with the mouse click technique, is the pop-up widget remains active only while the user hovers at the same point. Further, to select items within the widget, the user must be able to glide into the widget space without leaving its boundaries, else the widget disappears (the link has lost the mouse-over focus).

The HyperScout model uses the mouse-over to trigger a pop-up, and thus inherits the disadvantages of mouse-overs addressed previously. To overcome these disadvantages the *HvU* model includes a *thumbtack* metaphor. A thumbtack is used to tack open a pop-up. In this way, the user can have numerous pop-ups open and can easily make comparisons between links. The thumbtack appears in the top right corner of the pop-up, as can be seen in Figure 6.

4 Methodology

This paper explores a range of techniques for displaying additional link information, many of which have been incorporated into the *HvU* implementation or are planned for future versions. The general methodology is a progression from static to dynamic annotations, coinciding with a progression from author-specified annotations to proxy-generated annotations.

4.1 Static Annotations

Author-specified annotations are necessarily static. They exist in the page at the time of creation, i.e. before the page is requested. The first step in development emulates the browser's built-in annotation, followed by a ToolTip implementation, then a more substantial pop-up.

Currently, the browsers Netscape and IE5 display a link's target URL in the status bar (at the bottom of the browser window) when the link is moused-over. This step is implemented with a browser and HTML, tools that are available to any web author.

However, the status bar is ineffective in presenting information to a novice user. The user's attention is focused on the link, so they may not notice the activity at the bottom of the window. Thus, the next stage of development explores techniques for displaying additional information at the link. A basic ToolTip is developed.

The preceding steps only allow one line of text to be displayed. The development of annotations that are more substantial is next. These pop-ups contain any amount of text or graphics.

4.2 Dynamic Annotations

A browser, from the static content of a link anchor's `href` attribute dynamically generates the status bar URL. Other than this, current web browsers do not have the ability to offer any information on a link's target.

In the case of dynamically generated pages, such as those produced by a search engine, the annotation techniques described in the previous section could be dynamically included. Nevertheless, a goal of this paper is to define a solution that is immediately available to all users, without waiting for search engines and authors to improve their page design.

The next development stage dynamically inserts annotations. The annotations are implemented using either ToolTips or pop-ups. *Dynamic* in this context

refers to the modification of a page after leaving the server. This modification is carried out by either an intermediary application (a proxy) or the browser itself.

An intermediary that dynamically generates and inserts annotations already exists - HyperScout. Because of this, the remaining stages of development revolve around enhancing the *HvU* model and application.

The first step in the dynamic processes involved implementing those informational features not included in HyperScout, i.e. domain type and country information, and images corresponding with a link's MIME type.

This was followed by the addition of a user preferences system. Initially, the 'dummy' preferences file is local to the proxy, so each client of the proxy receives the same level of detail. Ideally, each browser has a local preferences file, so that *HvU* can access and satisfy an individual user's preferences.

The final phase is the development of the thumbtack metaphor.

5 Implementation

This section describes the development of the annotations. Beginning with a very basic, very limited, incarnation, the development progresses through increasingly complex and sophisticated annotations. Each annotation technique is also more dynamic than the last.

The static annotations of status bar URL, ToolTip, and Pop-up, were created using various combinations of HTML, JavaScript, and browsers (IE5 and Netscape 4.7). Results from the Kanoodle (Kanoodle 2000) search engine, which was chosen for the lack of information it presents to users, are used to illustrate the techniques. Other search engines present more information with their results, but are mostly limited to keyword highlighting, file size, and relevancy scores. Moreover, each search engine provides a different level of detail. *HvU* may still prove useful with other search engines, by providing moe detail and a consistent interface to the user.

The dynamic annotations required additional tools. WBI (IBM 2000) is a proxy implemented in Java, and is expandable through creating plug-ins, HyperScout being one such plug-in. HyperScout itself requires the MySQL database software and the ADC (1998) HTML parser package. The Java 2 development kit was needed to modify HyperScout.

5.1 Status Bar

The status bar value is set through the `window.status` attribute. Resetting the `window.status` value on an `onmouseover` event achieves the usual status bar behaviour, but with an author-specified value.

```
<a href="…" onmouseover="window.status
='http://www.auburn.edu/~vestmon/robot
ics.html'; return true;">
```

The above code fragment replaces the default status bar value (a particularly complex URL, Figure 1) with a more meaningful value (the eventual destination of the link, Figure 2).

5.2 ToolTip

With HTML version 4.0, web page authors can enhance link anchors by applying the generic attribute `title` to the anchor element `a`. Netscape 4.7 ignores the title attribute, thus IE5 or Netscape 6 is required to benefit from this feature. Figure 3 shows the appearance of the title ToolTip in IE5 (MacOS implementation). The Windows ToolTip is an elongated rectangle.

5.3 Pop-ups

JavaScript, in addition to HTML, was required to implement pop-ups. A JavaScript 'pop-up' is implemented as a table cell, whose visibility is toggled between visible and hidden. The toggle is triggered by `onmouseover` (visible) and `onmouseout` (hidden) events. An HTML table is defined for each link. As any HTML code may appear in a table cell, the amount of information that is displayable in the pop-up is not limited. This technique also has the advantage of working in both IE5 and Netscape.

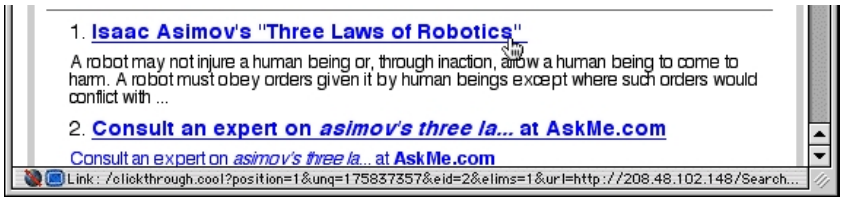

Figure 1: Original Status Bar

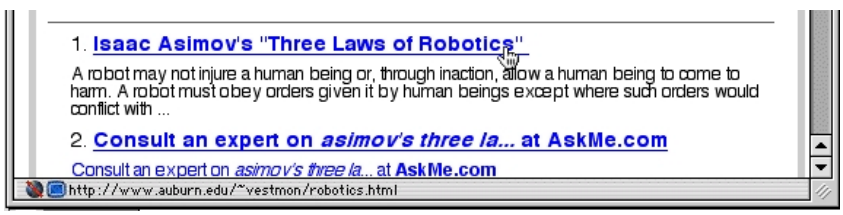

Figure 2: Improved Status Bar

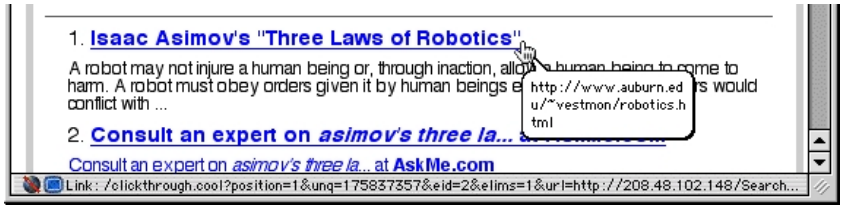

Figure 3: ToolTip

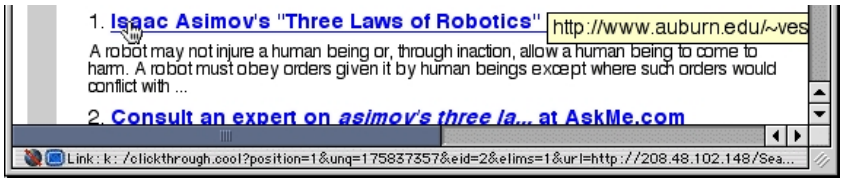

Figure 4: Pop-up

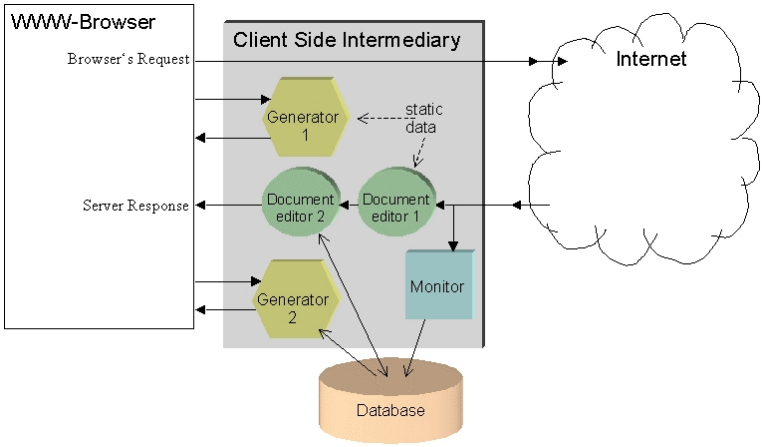

Figure 5: HyperScout Architecture — Client (Weinreich 2000)

There are two possible pop-up implementations. The simplest implementation inserts the table immediately following the anchor tag. To avoid the shortcomings of this approach, a second, more complex, solution is possible. This captures mouse events and binds the location of the pop-up to the mouse coordinates. The latter approach is used in HyperScout. Figure 4 illustrates the pop-up.

5.4 HyperScout

This section describes the implementation of Hyperscout. First a brief summary of the concepts behind WBI are discussed, followed by a detailed description of Hyperscout proper.

WBI has four modules, Monitor, (Document) Editor, Request Editor, and Generator, collectively known as MEGs. Request Editors receive a request and may modify the request before passing it along. Generators receive a request and produce a corresponding response (i.e. a document). Editors receive a response and may modify the response before passing it along. When all the steps are completed, the response is sent to the originating client. A Monitor can be designated to receive a copy of the request and response but cannot otherwise modify the data flow (WBI 2000). A collection of MEGs fulfilling some new functionality is called a plug-in. More detailed information is available from the WBI web site.

The client side implementation of HyperScout processes requests and responses in the following sequence. The browser request is sent directly to the Internet, i.e. HyperScout does not have a Request Editor. HyperScout intercepts the response, which is parsed by the Monitor and Document Editor 1. The page is forwarded to Document Editor 2 and Generator 2. Document Editor 2 passes the response onto the browser, while Generator 2 continues to append information to it. Generator 1 is responsible for some file transfers. Figure 5 describes the architecture of HyperScout.

The Monitor stores in the database the attributes for a retrieved page. Document Editor 1 inserts two links in the `<head>` of the page.

The first link is to a Cascading Style Sheet (CSS), and the second, a link to a JavaScript file. The CSS controls the appearance of the pop-ups. The JavaScript file contains the functions that control the actions of the pop-ups. Generator 1 transfers these files from the proxy's hard disk to the browser.

Document Editor 2 appends JavaScript commands to all `a` and `area` tags. A link is given a unique identifier. Calls to JavaScript functions are inserted after the `href` attribute, according to mouse events. That is, `onmouseover` calls `hs_activate`, while `onmouseout` and `onclick` call `hs_deactivate`.

If a link has existing JavaScript calls for these events, Document Editor 2 inserts the new calls before the existing ones.

Generator 2 is the MEG that dynamically creates a pop-up. An HTML table will eventually hold all the available attribute icons, attribute names, and their values. The following example call to a JavaScript function is inserted at the end of the page, the function's definition having been inserted at the beginning by Document Editor 1. The function `hs_newPopup` is responsible for creating a division with `linkid1` as its id attribute, and a table to hold the string of table rows and table data elements.

The server side HyperScout was not the focus of this study, but its architecture is presented here for completeness. The server side intermediary contains one MEG, a Document Editor that inserts additional attributes into anchor tags. These attributes are taken from the database maintained by the intermediary's robot.

5.4.1 HyperScout Extensions

HvU includes two additional attributes, the thumbtack metaphor, and a user configuration system. Two additional attributes *Country* and *Domain* are implemented with a new set of Java functions. The `getCountry` method extracts the two-letter country code (if it exists) from the target URL, and accesses the hash table `countryTable` for the corresponding country name. The `getDomainType` method extracts the three-letter domain code, and accesses the hash table `domainTable` for the corresponding domain type.

5.4.2 Thumbtacks

Initially, the thumbtack was implemented to function with the simple static pop-ups created earlier. The modified `hide` function tests the tack status of a pop-up. If the link's id is found in the tacked array, the pop-up remains visible, otherwise it is hidden. A pop-up is tacked by clicking the tack image in the top right corner of the pop-up. This calls the JavaScript function `tack`, which inserts the link's id into the array. Clicking the tack image again removes the id from the array.

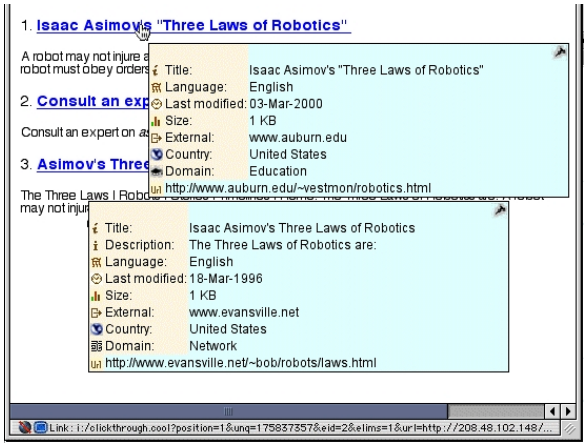

Figure 6: Overlapping HyperScout Pop-ups

Implementing the thumbtack for *HvU* was more complicated. Clicking the tack image requires a user to move the mouse into the pop-up table area. Doing so triggers the `mouseout` event, thus deactivating the pop-up before the tack can be clicked. In order for the user to move into the pop-up, the call to `hs_deactivate` was removed from the `onmouseout` event. Tracking the mouse coordinates caused the next difficulty. The mouse-move event was assigned to the function `hs_checkMouseMove`. Moving the mouse into the pop-up area caused the pop-up to move with the mouse. Thus, it is impossible to 'catch' the pop-up. However, this bug could later be used to implement drag-and-drop, allowing the user to re-locate a pop-up. A successful implementation would need to determine the area of the screen a pop-up occupied, and allow the mouse to be moved into this area without triggering a mouse-out event. Figure 6 shows two *HvU* pop-ups, the lower one having been tacked open before the upper pop-up was activated.

5.4.3 User Configuration System

The user configuration system is implemented by including a condition test for each link attribute. If an attribute is set to ON, the attribute is included in the pop-up. If an attribute is set to OFF, the attribute is ignored. A new method, `getUserOptions`, controls the appearance of icons and attribute names within a pop-up. This method tests the status of the user's icon and name configuration, and calls the appropriate method. Two new methods handle the cases when: no icon or name is required (only the value); only the icon (and value) is required. An existing method was sufficient for the case of only the name and value displayed.

The user configuration system also includes a graphical user interface. While not an essential component of the *HvU* model, it is nonetheless helpful in modifying the `hyperscout.properties` file.

6 Evaluation

This section consists of an academic review of *HvU*. It was found that the link annotations work in principle, and can be implemented through a number of methods. Naturally, the degree of usability must be established through useability testing.

During the implementation stage, it was discovered that the methodology had two further progressions that resonated with the static to dynamic development. The number of tools required was proportional to the dynamic level being implemented. Implementing the static status bar required only a browser and HTML, but implementing dynamic pop-ups required HTML, JavaScript, a proxy (WBI), mySQL database, and an HTML parser.

The other correlation was between the dynamic level, and the amount and accuracy of information presented. Consider the modification date of a node. Included statically by the author of a referencing page, this datum

is likely to become obsolete quickly without frequent checks by the author. Contrastingly, a dynamic inclusion of this datum ensures its correctness.

While a modification date is either correct or incorrect, other data such as keywords and file size have degrees of accuracy. For example, consider a set of keywords *Y* that describes a recently modified page (not the *recorded* set of keywords), and a recorded set of keywords *X* applicable to either the previous version/s or the current version of the page. An author modifies a page and neglects to update the set of keywords. *X* is the set the author forgot to update, and *Y* is the set of keywords that actually describe the page. *Y* differs from *X* such that:

- Y includes new keyword elements $y \in Y, y \notin X$
- Y does not include some old elements $x \in X, x \notin Y$
- Y retains some of the same elements $z \in X, z \in Y$.

Thus, *X* is an accurate set of keywords if it is equivalent to *Y*

$X \subseteq Y$ and $X \supseteq Y$ (i.e. $X = Y$).

X is inaccurate if some of the keywords in *X* are in *Y* and some of the keywords in *X* are not in *Y* (or vice versa). Such that $X \cap Y = Z$, where

$Z \neq \varnothing$, and

$Z \subseteq X$ and $Z \subset Y$, or

$Z \subseteq Y$ and $Z \subset X$.

X is completely incorrect if none of the keywords in *X* are in *Y*

$X \cap Y = \varnothing$.

If an author always updates the keyword list whenever a change is made, then the keyword set *X* is always equivalent to *Y*, meaning the recorded keyword set is completely accurate. If the author alters a page without updating the keywords *X* to match *Y*, the recorded keywords are either partially incorrect or completely incorrect.

Given a completely dynamic implementation, *X* will always be equal to *Y*, thus producing the highest level of accuracy. As the determination of attributes becomes less dynamic, $X \cap Y$ becomes smaller. Thus, a dynamic implementation is the only way to guarantee accuracy.

The different annotation techniques were useful to varying degrees of success. A brief criticism is offered of each technique.

6.1 Status bar

Manipulating the status bar is the simplest technique that works in both Netscape and IE5. It is therefore available to all web authors and web users. Applying this technique to the Kanoodle search engine, the benefits of this technique are clear. The URLs returned by Kanoodle are quite complex (they pass through a portal before arriving at the real target page). Figure 1 shows the status bar display for one of these links. By setting the status bar to the real target, Figure 2, the user has a better understanding of where they are going.

However, the annotation is affected by the size of the window. Much of a long annotation will not be visible when the window is very small. Even if the window is full-size, an annotation may contain more characters than can be displayed.

This problem is overcome by coding a *scroll* mechanism. The annotation appears at the right end of the status bar and appears to scroll left. The implementation essentially concatenates a string of spaces with the annotation, forcing the annotation to the right. Then, the status bar value is continually replaced by the previous string minus the first character, making the annotation scroll left. While allowing the entire annotation to be displayed, users may miss the beginning. Even if it is looped, information presented in the annotation will be difficult to synthesise, especially if the scrolling is too fast.

6.2 ToolTip

A ToolTip is superior to the status bar in that the information is displayed at the point of the users attention. Also, the amount of text that can be displayed is not limited. The ToolTip adjusts its size to cater for the text.

Unfortunately, the content of the title attribute cannot be formatted, nor can it include graphics, both of which an author may wish to use when annotating their links. Moreover, only users with IE5 or Netscape 6 benefit from this technique. Another disadvantage is the inconsistent appearance across platforms. On MacOS, a ToolTip appears as a balloon (or speech-bubble), while on Windows it is a rectangle.

6.3 Pop-up

Pop-ups have several advantages over ToolTips. Pop-ups can contain images and formatted text; have a consistent appearance across platforms; and they work in both browsers.

However, the simple implementation of pop-ups does not behave as a user might expect. The pop-up table is defined within the link itself, between the `<a>` tags. When its visibility is toggled, the pop-up appears in the same location where it is defined. That is, it is fixed to appear at the end of the link anchor text. Figure 4 shows the mouse at the far left of the link while the pop-up appears at the far right. Additionally, this implementation does not detect the edge of the window, so part of the pop-up is hidden from view.

The major disadvantage of pop-ups is their increased size. The more information contained in a pop-up, the larger the area surrounding the link is obscured.

6.4 HyperScout

HyperScout detects the edge of the window and repositions the pop-up accordingly. This is necessary, as HyperScout pop-ups can be quite large. Unlike pop-ups, this effect is minimised in *HvU* by incorporating the user

configuration system. Users can chose a minimal number of attributes to be displayed, as well as turning off the space consuming icons and attribute names.

Many of the icons used by HyperScout are not immediately interpretable, for instance, the *Language* and *Reference* icons. In particular, the icons for *Author*, *Description*, and *Title*, are similar in appearance (Figure 7). These icons need redesigning, and need to undergo user evaluation before being included in an official release.

Figure 7: HyperScout icons - Language, Reference, Author, Description, Title

The thumbtacks are advantageous when users wish to compare two or more links. In Figure 6, the bottom pop-up is tacked open before the top pop-up is activated. The pop-ups are very near to overlapping. If two links are near each other on a page, the pop-ups are likely to overlap and one annotation will be obscured. The drag-and-drop mechanism, discovered in Section 5.4.2, could be used to alleviate this problem. However, as the user moves a pop-up away from its link, there is no indication (other than page title and URL) to which link the pop-up belongs. Even in its original location, a pop-up is not actually connected to its link. If many pop-ups are tacked open, it may become confusing to which links pop-ups belong. The MacOS 'balloon' technique (with its pointer to the source) would be a useful enhancement to the pop-up.

Stanyer and Procter (1999) raised the question of how much information is enough information. The opposite question, how much is too much, is also relevant when annotating links. By including a user configuration system, the user can indicate the level of detail they require, as they are in the best position to know what they want.

7 Conclusion

This paper shows that a number of techniques are available for annotating hyperlinks. It found that annotations of differing complexity and usefulness are achievable.

The usefulness of an annotation relies greatly on when it is created. An annotation created by an author, though assisting the author in explaining the intention of their link, is limited to presenting static data, i.e. information that is unlikely to alter over time (such as the title). Given the fluctuating nature of the web, a user needs the most current information available on the changeable aspects of the web, such as server and page status. The only way to do this is by dynamically generating annotations.

Thus, priority should be given to the development of dynamic annotation generators, such as HyperScout. Given that the ideas presented here were implemented with existing tools, there seems little reason why current web servers and browsers cannot incorporate a mechanism for assisting users in their navigational experience.

8 References

AKSCYN, R., MCCRACKEN, D. and YODER, E. (1987): KMS: A Distributed Hypermedia Systems for Managing Knowledge in Organizations. *Communications of the ACM*, **31**(7):820-835.

ADC (1998): HtmlStreamTokenizer. [Online] Available : http://www.do.org/products/parser/ [Accessed 25 July 2000].

CHEN, C. (1999): Visualising semantic spaces and author co-citation networks in digital libraries. *Information Processing and Management*. **35**:401-420

CONKLIN, J. (1987): Hypertext: An Introduction and Survey. *IEEE Computer*, **20**(9):17-41.

IBM (2000): WBI (*Web Intermediaries*). [Online] Available : http://www.almaden.ibm.com/cs/WBI/ [Accessed 25 July 2000].

KANOODLE (2000): [Online] Available : http://www.kanoodle.com/ [Accessed 25 July 2000].

KOPETZKY, T. and MÜHLÄUSER, M. (1999): Visual Preview for Link Traversal on the WWW. *8th International World Wide Web Conference*, Toronto, Canada. [Online] Available : http://www8.org/ [Accessed 25 July 2000].

LANDOW, G. (1991): The Rhetoric of Hypermedia: Some Rules for Authors. In *Hypertext, Hypermedia and Literary Studies*. DELANY and LANDOW (eds). MIT Press.

LOEB, S. (1992): Delivering Interactive Multimedia Documents over Networks. *IEEE Communications Magazine.* May:52-59.

SCHNASE, J., LEGGET, J., HICKS, D. and SZABO, R. (1993): Semantic Data Modelling of Hypermedia Associations. *ACM Transactions on Information Systems*. **11**(1):27-50.

STANYER, D. and PROCTER, R. (1999): Improving Web Useability with the Link Lens. *8th International World Wide Web Conference*. Toronto, Canada. [Online] Available : http://www8.org/ [Accessed 25 July 2000].

WBI (2000): *The WBI Development Kit Documentation*. [Online] Available: http://www.almaden.ibm.com/cs/wbi/ [Accessed 25 July 2000]

WEINREICH, H. and LAMERSDORF, W. (2000): Concepts for Improved Visualisation of Web Link Attributes. *9th International World Wide Web Conference*. Amsterdam. [Online] Available : http://www9.org/ [Accessed 25 July 2000].