Improving Target Acquisition in Web Applications with Link Prediction

by

Roger B. Hanna

S.B., Massachusetts Institute of Technology (2005)

Submitted to the Department of Electric Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electric Engineering and Computer Science
at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

March 2007

©2007 Roger B. Hanna. MMVII. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole and in

Department of Electrical Engineering and Computer Science

March 12, 2007

Certified by

Robert C. Miller
Professor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

BARKER
Improving Target Acquisition in Web Applications with Link Prediction

by

Roger B. Hanna

Submitted to the Department of Electrical Engineering and Computer Science on March 12, 2007, in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Science and Engineering

Abstract

Web application users spend considerable time clicking on hyperlinks and buttons to complete frequent tasks. Individual application developers can optimize their interfaces to improve typical usage; however, no single task model will accurately reflect the needs of a wide audience of users. This thesis describes EasyLink, an automated optimization to the view of Web applications. EasyLink facilitates the common activities of individual users without explicit customization by each user. Using a record of the user’s actions, EasyLink adapts the view of the page on later visits. The new view reduces unused elements by decreasing their contrast and emphasizes the most used elements by enlarging their size and ease of pointing. An evaluation of EasyLink on Gmail shows that it accurately models 64% of user behaviour, significantly reduces the time to complete simple tasks, and is preferred by users over the default view of Gmail.

Thesis Supervisor: Robert C. Miller

Title: Associate Professor
Acknowledgements

I'd like to thank all the people who have given me their support and patience throughout my thesis: Professor Miller, Val, Kurt, Al, mom, and dad. I don't think I could have possibly divided my time further. I'd also like to thank the other students in the User Interface Group especially Greg, Kevin, Darris, Vikki, everyone who participated in my user studies, and everyone else who gave me feedback about my project. You rock.
## Contents

1 Introduction ................................. 15

2 Related Work ............................... 21

2.1 Target Acquisition Index of Difficulty ...................................... 21

2.1.1 Area Cursor ................................... 22

2.1.2 Expanding Targets .................................. 23

2.1.3 Jumping to Targets .................................. 23

2.1.4 Dynamic Control Display Gain .................................. 24

2.2 Modifying Web Page Content ........................................ 25

2.2.1 Automation ........................................ 25

2.2.2 Accessibility ....................................... 26

2.3 Link Prediction ........................................ 26

2.3.1 Assisted Searching ....................................... 27

2.3.2 Reducing Latency ....................................... 27

2.3.3 Design Evaluation ....................................... 28

3 Modeling User Tasks .......................... 29

3.1 Design Goals ....................................... 29

3.2 Element Pruning ....................................... 30
3.3 Feature Selection ................................................................. 32
3.4 Recording ................................................................. 37
  3.4.1 Listeners ................................................................. 37
  3.4.2 Content Selection .................................................. 37
  3.4.3 Scalability ............................................................... 38
3.5 Distribution Space ......................................................... 39

4 Gmail Task Study ............................................................... 41
  4.1 Method ................................................................. 42
  4.2 Results ................................................................. 43
  4.3 Discussion ............................................................... 43

5 Browser User Interface ...................................................... 47
  5.1 Visual Changes .......................................................... 48
    5.1.1 Font Size .......................................................... 48
    5.1.2 Contrast ............................................................ 49
  5.2 Mouse Controller .......................................................... 51

6 Look and Feel Study .......................................................... 53
  6.1 Method ................................................................. 53
  6.2 Results ................................................................. 56
  6.3 Discussion ............................................................... 59

7 Conclusion ............................................................................. 63
  7.1 Contributions .............................................................. 63
  7.2 Further Work ............................................................... 64
    7.2.1 Full Evaluation .................................................... 64
7.2.2 Additional Features .................................................. 65
7.2.3 Additional Web Application Evaluations ....................... 66

A Task Model Interface ..................................................... 69
B Enhancement Module Interface ....................................... 71
C UI Study Pages ............................................................. 73
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Simplifying Gmail</td>
<td>16</td>
</tr>
<tr>
<td>1.2</td>
<td>Evolution of Hotmail.com Login</td>
<td>17</td>
</tr>
<tr>
<td>1.3</td>
<td>Modified Hotmail.com Login</td>
<td>18</td>
</tr>
<tr>
<td>1.4</td>
<td>EasyLink Design Diagram</td>
<td>20</td>
</tr>
<tr>
<td>2.1</td>
<td>Bubble Cursor</td>
<td>22</td>
</tr>
<tr>
<td>2.2</td>
<td>Semantic Pointing</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>Visualizing Element Weights</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>Weights with Element Pruning</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Weights with Unread Class</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Weights with Ordering</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Weights without Text Content</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Training Model Evaluation</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>Click Sequence Minimum Threshold</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Click Sequence Maximum Threshold</td>
<td>44</td>
</tr>
<tr>
<td>4.4</td>
<td>User Differences</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>Size Module View</td>
<td>49</td>
</tr>
</tbody>
</table>
5.2 Contrast Module View .................................................. 50

6.1 Look and Feel Study: Introduction .................................. 54
6.2 Look and Feel Study: Entire Login Screen .......................... 56
6.3 Look and Feel Study: Task Exit Questions .......................... 57
6.4 Spam Task Results ..................................................... 58
6.5 Reply Task Results ..................................................... 58
6.6 Label Task Results ..................................................... 59
6.7 Qualitative Results ..................................................... 59

7.1 Google Calendar Data ................................................... 67

C.1 UI Study Spam Description ........................................... 73
C.2 UI Study Reply Description .......................................... 73
C.3 UI Study Label Description .......................................... 74
C.4 Study Spam Page 1 ..................................................... 74
C.5 Study Spam Page 2 ..................................................... 75
C.6 Study Spam Page 3 ..................................................... 75
C.7 Study Reply Page 1 ..................................................... 76
C.8 Study Reply Page 2 ..................................................... 76
C.9 Study Reply Page 3 ..................................................... 77
C.10 Study Reply Page 4 ................................................... 77
C.11 Study Reply Page 5 ................................................... 78
C.12 Study Label Page 1 ................................................... 78
C.13 Study Label Page 2 ................................................... 79
C.14 Study Label Page 3 ................................................... 79
List of Tables

1.1 Hotmail Complexity Time Line .................................................. 17

2.1 Chickenfoot Trigger to Automate Hotmail Login ......................... 26

3.1 Recorded XPath ........................................................................... 36

4.1 User Data Size ........................................................................... 43

5.1 Mouse Controller Pseudocode ..................................................... 52

6.1 Task and Variant Data Size ......................................................... 56

6.2 Task and Variant Error Rates ....................................................... 57
Chapter 1

Introduction

Web pages are cluttered with content, navigational toolbars, and advertisements. Web browser improvements have enabled developers to provide Web-based services such as Webmail, eCommerce, blogs, and discussion forums. However, not all parts of a Web page benefit individual users. Excess content hinders users from finding and clicking on frequently used parts of the page.

This thesis introduces EasyLink, a Firefox extension designed to simplify common tasks in Web applications. After identifying which elements the user interacts with, EasyLink improves the usability for frequent tasks by sacrificing usability for infrequent tasks. Consider Gmail, a Web-based email application (Figure 1.1a). Its main page has roughly a hundred clickable elements, but a typical user only wants to read new messages. EasyLink takes this knowledge and makes that specific task easier and faster (Figure 1.1b). Unread messages are visually enlarged and the mouse pointer is slowed when it is over these messages; other messages are faded into the background to simplify the overall view of the Web page.

Simplifying a Web page’s view is critical as page complexity continues to increase over time. Using the Wayback Machine [26], consider the progression of Hotmail’s login page. What once
Figure 1.1: Views of Gmail: (a) default view, (b) simplified view presented by EasyLink, emphasizing unread messages
was a sparse login page is now filled with much more than a form with username and password (Figure 1.2). As shown in Table 1.1, both the size of the page and the number of clickable elements have grown over the years, making the most common task of logging into an account increasingly difficult. An HTML-savvy user may notice this and invest the time to set up an individualized entrance page with just the necessary fields. However, the individualized approach requires delving into the page source to extract form data and may fail when Hotmail updates its login page again.

![Figure 1.2: The evolution of Hotmail.com's Login Page](image)

<table>
<thead>
<tr>
<th>Year</th>
<th>Minimum Page Size</th>
<th>Clickable Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>640x480</td>
<td>49</td>
</tr>
<tr>
<td>2002</td>
<td>800x600</td>
<td>63</td>
</tr>
<tr>
<td>2006</td>
<td>1024x768</td>
<td>93</td>
</tr>
</tbody>
</table>

Table 1.1: Hotmail Complexity Time Line
EasyLink handles the Hotmail login intelligently, without necessitating expert knowledge or even user awareness. Using the information that a user only interacts with the login fields, EasyLink improves usability by enlarging important features and minimizing the clutter (Figure 1.3). The username and password fields have increased font size, while the rest of the page is muted.

![Modified Hotmail.com After Personal Training.](image)

The benefit produced by EasyLink is not limited to login pages. EasyLink assists with all common Web application activities by bringing simplicity back to Web pages. EasyLink adjusts what users see by emphasizing elements they interact with and reducing advertisements, extraneous features, or any other elements they do not click on.

The design of EasyLink is based on three principles: Automation, Invisibility, and Agency.

**Automation**

EasyLink handles customization of Web pages autonomously, giving end users the strength of the system without the cost of understanding it. Thus it can micromanage Web applications to optimize many more targets than a user would care to customize by hand.
Invisibility

Extensive visible changes to a Web page are distracting. If EasyLink creates too much distraction, it will discourage people from using it. Ideally all EasyLink’s changes would be subtle. Even if they are noticed, changes should not irritate the user.

Agency

The most important aspect of EasyLink is that all options remain available to a user. Completely removing an element from its Web page or jumping the mouse pointer over any target is not acceptable. Because users’ tasks are likely to change, removing any option should be avoided at all cost.

EasyLink’s design can be divided into two main parts: (1) the user task model which records input events and suggests a likelihood of interaction for an element; (2) an automated look and feel extension which uses model predictions to reduce the contrast of unlikely elements or both increase font size of likely elements and slow the mouse pointer over these elements. An overview of EasyLink is diagrammed in Figure 1.4. When a user visits a Web page, the user task model loads prior knowledge about the use of that page and passes the probability distribution for the elements to the look and feel modules that alter the user’s view.

EasyLink was developed in Firefox because of the strong division provided between the internal representation and the rendered view of a Web page. Gmail was the primary Web application considered because the complexity of its design exceeds most other Web applications. These techniques could be applied to any application or user interface toolkit.

The rest of this thesis is divided into six chapters. Chapter 2 discusses prior work on mouse usability, Web accessibility and customization, and machine learning in Web browsing. Chapter 3 discusses the design and implementation of the user task model. Chapter 4 evaluates the user task model. Chapter 5 discusses the design and implementation of the look and feel modules.
Figure 1.4: EasyLink Design Diagram

Chapter 6 evaluates the look and feel modules. Chapter 7 discusses the contributions of this thesis and the implications for future research and development.
Chapter 2

Related Work

Prior related work is focused on three different areas. They are: (1) improvements to time spent moving to clickable targets; (2) browser customizations; or (3) web usage mining to assist with searching, to reduce browsing latency, or to improve existing designs.

2.1 Target Acquisition Index of Difficulty

Target acquisition time has been modeled by Fitts’ law (Equation 2.1) since the seminal studies by Fitts [13] and MacKenzie [20]. Fitts’ law states that movement time (MT) is constrained by the amplitude of the movement (A) and the target’s width (W). Other environmental conditions affect the empirically determined constants, $a$ and $b$. The logarithmic term of amplitude over width is referred to as a target’s index of difficulty (ID).

$$MT = a + b \log \left( \frac{A}{W} + 1 \right)$$  \hspace{1cm} (2.1)

Researchers have investigated several different ways to improve the index of difficulty for pointing tasks. Proposed solutions are to increase the target width with an area cursor or expanding targets, to decrease the amplitude by moving the pointer and object closer together, to
improve both amplitude and width by dynamically adjusting the control display gain.

2.1.1 Area Cursor

An area cursor increases the effective width of a target by using a region of activation rather than a point of activation as in traditional pointing. Kabbash and Buxton [17] proposed the original system and showed this system could be modeled by Fitts’ law. However, increasing the activation area of the cursor posed a problem: how should the area cursor resolve activation conflicts when two or more targets are within the region? Worden et al [27] resolved the problem by including a single point of activation when more than one target is in the area of activation. This enhanced area cursor gave the benefits of the original system with the authoritativeness of traditional pointing.

Grossman and Balakrishnan [14] proposed another improvement with their bubble cursor. Like the previous area cursors, a bubble cursor activates a region, but the region dynamically resizes so that exactly one target is within the activation region. Figure 2.1 divides the resulting clickable regions with dashed grey lines for each circular target. Elements have an enlarged clickable area, with no impact on the application’s layout nor the cursor’s movement. As a result, the bubble cursor makes efficient use of the provided space with no effort by the application’s developer.

![Figure 2.1: Bubble Cursor Activation Regions [14]](image)

In a complex application that maximizes its clickable area (such as Gmail), the extra white
space that is required by the bubble cursor is not available. Thus the bubble cursor degenerates to traditional pointing. For example, suppose the user wants to select a character within a paragraph. Since every character of text is a potential target, the bubble cursor cannot improve on the original index of difficulty.

2.1.2 Expanding Targets

Another means to increase the width of a target is to expand the target's actual size as the mouse pointer approaches the target. McGuffin and Balakrishnan [21] showed that users benefit from expanded targets after 90% of the distance had been traveled. Zhai et al [29] determined that expanding targets had the expanded target's index of difficulty. While expanding targets improve a single target's index of difficulty, it has not been shown to be effective for multiple nearby targets, which includes the most well-known implementation, the Mac OS Dock.

2.1.3 Jumping to Targets

Many displays have sparse sets of targets. Rather than adjusting the width of the targets, Baudish et al [5] bring the target to the cursor with the drag-and-pop technique. By examining the direction of movement, drag-and-pop can identify hard to reach targets and provide alternate, closer aliases. This system is designed for large displays where it would be too cumbersome to move the entire distance.

Most computer systems don't have the physical distance problem drag-and-pop resolves. Another alternative is to bring the cursor to the target. Guiard et al [15] designed a system to skip over void space so that the cursor is always pointing at an object. Both of these techniques are a dramatic change from traditional pointing on a displayed screen image. Drag-and-pop requires users to hit a previously non-existent target and object pointing has no cursor at all. These kinds of changes are not feasible as incremental improvements to existing computer systems.
2.1.4 Dynamic Control Display Gain

The final way to improve movement time with respect to width and amplitude is to intelligently change both values. By adjusting the control display (CD) gain (the ratio of control device movement to displayed position movement), both Keyson [18] and Worden et al [27] improved movement time. If the cursor is outside the expected target, an increase in CD gain reduces the movement's amplitude. If the cursor is inside the expected target, a decrease in CD gain increases the target's width. By combining these two ideas, targets intended to be frequently used can have reduced movement times without requiring a change to the appearance of the page or a change to the appearance of the cursor.

Blanch et al [7] proposed an extension to user interface toolkits to specify a widget’s CD gain. In this system, called Semantic Pointing, the mouse’s speed changes relative to the element which it is over. Semantic pointing allows specific targets to have a smaller movement time, relative to other targets. In effect, a user interface designer controls the visual space independent of the motor space. For example, while a user may see a consistently sized view of a menu (Figure 2.2a), the mouse cursor will pass over unusable menu items more quickly (Figure 2.2b). Applied within a browser, this system of weighting elements’ motor space emphasizes targets that are expected to be used frequently by making their effective width larger. Additionally, dynamically scaling mouse speed is not obvious to the user, even though users benefit from it.

Figure 2.2: Semantic Pointing: (a) Visual Space, (b) Motor Space [7]
2.2 Modifying Web Page Content

Different users interact differently with Web pages; it is not advantageous to have all users view the same page. Fortunately, Web pages only suggest what a user should see, and the user's browsing environment determines the rendered view. Early work such as Barrett et al's WBI [4] demonstrated that client-side modification of the view was an effective tool to personalize Web pages. There are three components to Web page personalization: automating the Web browser, improving accessibility, and simplifying the displayed content.

2.2.1 Automation

Initial work to automate web browsers, such as WBI [4] and Greasemonkey, required intimate knowledge of a web page's source in order to customize the Web page. Since most users don't have this knowledge, simpler interfaces were necessary. Anupam et al [1] enabled users to automate tasks with their WebVCR. Users could record sequences of clicks and replay these macros at a later time. Because users had to demonstrate each situation explicitly, WebVCR had limited its scope to automating simple tasks.

To handle a broader set of tasks, parameters become a necessary tool. Bolin et al [8] provided a simple interface to arbitrarily automate a Web page for end users. Their system, Chickenfoot, provided its own set of understandable functions to operate on a Web page independently of the underlying representation. For the Hotmail login page from Chapter 1, Chickenfoot can automate the process of logging into Hotmail without requiring knowledge of HTML (Table 2.1). End users can gain some of the performance benefits of EasyLink with Chickenfoot, but they have to recognize the problem and spend the time to resolve it.
Table 2.1: Chickenfoot Trigger to Automate Hotmail Login

2.2.2 Accessibility

Not all users can effectively use the typical view of the Web. Older adults have degraded vision and movement precision. Disabilities also may hinder Web page usability. Focused work had been done targeting individual handicaps, such as text-to-speech for blindness [3], [28] or increased visibility for low vision [9]. Most designs were limited to a group of users and to a subset of Web pages. Richards and Hanson [23] approached Web browser enhancement as a means to improve the overall accessibility of a Web page for all users. They provided a broad set of enhancements so individual users could pick and choose which ones they preferred. These enhancements included audibility improvements (text-to-speech for both text and image ALT tags), readability improvements (enlarged font size, spacing, and contrast), and complexity reduction (hiding animations, backgrounds and images). With this approach to the problem, Richards and Hanson were able to improve Web page usability for a broad audience of users.

Approaching accessibility as a generalizable, robust solution is an important concept that carries over to EasyLink. Since many Web applications have good color schemes, we don’t want to discard this information. To best preserve both Web application style and Web accessibility, EasyLink only blends existing colors and increases the size of text.

2.3 Link Prediction

Link prediction and data usage mining are machine learning techniques used to identify what a user interacts with in a web page. Use of these techniques enables developers to improve a
user's web browsing experience based on prior knowledge of the user. This can be done on the user's computer (client-side) to reduce latency, on the server to improve caching, and off-line by developers to improve the page design. Initial work in link prediction came out of the need to facilitate searching on the Web.

2.3.1 Assisted Searching

Armstrong et al [2] designed an extension, WebWatcher, to help users locate information. It asynchronously fetches pages with content similar to what the user requested, and alters its recommendations based on explicit user feedback. WebWatcher helped inexperienced users find content on their subject matter and it helped advanced users do so more quickly. However, WebWatcher fails when people are unaware of precisely what they want.

Lieberman [19] addressed this problem with Letizia by looking for similarities in the pages the user browsed. By suggest alternate pages with similar content, Letizia trained its user model implicitly on whether users continued with the recommended pages or went back to a prior Web page. Both WebWatcher and Letizia were interactive extensions for Mosaic with front-end changes to the browser, but their primary purpose was to actively assist in user searches.

2.3.2 Reducing Latency

As well as providing suggestions to the user, link prediction can be used to improve Web browsing performance. Bestavros [6] used trends in server HTTP requests to predictively cache and send additional pages with the requested page. Padmanbhan and Mogul [22] applied similar techniques on the client-side. Using knowledge of prior Web browsing sessions, they requested additional Web pages the user was likely to click on. Since users had to wait for Web page downloads less often, users browsed the Web faster.

This idea of invisibly facilitating expected links plays an important role in EasyLink, but
predictively caching additional web pages has limitations. It can not capture all generated content and it increases the total bandwidth every time it incorrectly predicts a link. Instead, EasyLink uses link prediction to improve a Web page's accessibility.

Another key difference between these systems and EasyLink's user task model is the type of data being learned. Existing link prediction systems operate on visited URLs to predict subsequent URLs the user will visit. Since Web applications don't necessitate URLs or even server requests, EasyLink cannot assume URLs are the only relevant information on a Web page. EasyLink operates on elements within the Document Object Model (DOM) to predict subsequent elements the user will click on.

User task models can be developed independent of their application. Other researchers proposed many subsequent improvements to the task model by using Markov Models [10], [12], [24], or page content [11].

2.3.3 Design Evaluation

Link prediction is not limited to on-line performance. Spiliopoulou [25] analyzed server HTTP requests off-line to identify common paths a user takes. This system enables developers to critically evaluate existing Web site designs and justify changes to the user interface.
Chapter 3

Modeling User Tasks

Measuring the way a user interacts with a Web page is a key feature of EasyLink. A probability distribution of the user’s interaction with elements within a Web page is necessary to make intelligent alterations. This chapter addresses the design of a model of the user’s interactions and noteworthy issues I encountered implementing it. The user task model is broken down into four parts: (1) selecting which elements to operate on; (2) representing the set of elements; (3) choosing when to train an element; and (4) providing a metric of user interaction.

All task model details are demonstrated as in Figure 3.1, which displays the frequency an element is clicked on as shades of grey. These frequencies, or element weights, are relative to others, so their average is 50% brightness. Darker values indicate elements that are used more frequently than others.

3.1 Design Goals

Before describing the details of the task model, the overall design goals are presented.

Automation

The task model needs to automatically prepare itself in advance so that it can to respond
in real time to whatever queries the enhancement modules make. EasyLink provides the task model a programming interface to enable this functionality (Appendix A).

### Invisible

EasyLink should have invisible long term data storage. It should use a negligible amount of disk space and should minimize the amount of stored confidential or personal information.

### Agency

All of EasyLink’s computation needs to be done in real time. Spending 200 milliseconds may be an acceptable cost when a document is first loaded, but no noticeable delay is acceptable in response to a mouse or keyboard event.

#### 3.2 Element Pruning

EasyLink operates on a subset of elements in the DOM. Pruning the element set reduces model complexity and provides a more consistent view to the user. For example, the simplest Gmail
inbox has roughly 800 HTML elements, creating 120 uniquely clickable objects. The other 680 elements provide stylistic layout to the page by emphasizing parts of the information such as unread subjects, and deemphasizing less important parts like the message body. The difference in user interaction models resulting from element pruning is demonstrated in Figure 3.2. Without element pruning, the message label, subject, and body each have different weights. With element pruning, they have a uniform weight because they are operated on as a group.

![Figure 3.2: Weighted Elements](image)

EasyLink operates on one clickable element and its non-clickable descendants as a group. Unfortunately, current Web browsers do not provide the necessary mechanism for EasyLink to determine what elements actually have click event handlers attached to them. EasyLink solves this problem by heuristics that derive semantic meaning from syntax. Syntactically clickable elements have one of these properties making them relevant to the task model: (1) they are a block element (an element that is rendered as a rectangle) which developers are likely to use for grouping purposes, or (2) they are an inline element (an element that does not form a new block) with a known native click event handler (such as `<A>` or `<INPUT>`). The block elements are significant because they naturally group descendents together in the rendered view, making them appear as a single object to the user. The inline elements are significant because they always capture click events. The complete list of syntactically clickable elements is: `<A>`, `<APPLET>`, `<BUTTON>`, `<DIV>`, `<INPUT>`, `<LABEL>`, `<OPTION>`, `<SELECT>`, `<TD>`, `<TEXTAREA>`, `<TH>`. 
3.3 Feature Selection

EasyLink needs a means of identifying the same element across browsing sessions. Even with the reduced set of clickable elements, a plethora of features remain to identify elements. I have considered several feasible features to observe: tag name (the type of the element), class name (the style of the element), previous target (the previously clicked element), child index (the element’s position in the DOM), node control name (the label assigned by developers), and the element’s text content. There may be other features influencing a user’s decision, but there is a tradeoff between information gain and training rate loss as more features are observed. Unless the existing feature set is insufficient or less concise than other features, it is not necessary to consider other features.

Tag Names indicate which type of element is involved. Different types of elements (such as buttons, text fields, hyperlinks, and tables) look and behave differently, so a user would not consider them to be the same.

Class Names indicate the decoration of a element. Cascading style sheets (CSS) describe the style associated with each class name. Developers use CSS and class names to produce a consistent view across multiple documents and elements without transferring excessive, redundant information from the server to the client. Distinct class names suggest different element states that may encourage or discourage a user from clicking on that element. For example, Gmail changes the class name of table rows to indicate whether a message is unread or not. The difference in what the user clicks on resulting from different class names is demonstrated in Figure 3.3. Unread messages and read messages have different class names. Since a user is more likely to click on an unread message than a read message, unread messages are weighted higher.
Previous Target is the element the user previously clicked on. At times, users must perform a sequence of clicks to accomplish a single task. For example, in Gmail you can tell when you have unread messages in a folder. These messages can only be read after clicking on that folder, followed by a click on the message. To distinguish between independent tasks, only consecutive clicks within five seconds are considered. Five seconds should be enough time to perform any planned movement. This cut off makes it possible to discard information when the user pauses to read or think.

While the previous target is an important feature, a non-distracting static view of the page is preferred for its invisibility than the potential reduction of the task's index of difficulty resulting from dynamically rescaling elements. Therefore, the previous target is only used for invisible changes to the Web page's look and feel: adjusting the mouse speed.

Examples of Gmail tasks with sequential clicks include:

- Filing multiple messages

One of Gmail's main filing operations is to first select a set of messages by clicking on their checkboxes, and then to click on the appropriate filing button, be it archive, report spam, delete, or one of the many other options in the drop-down menu. When selecting several messages, people tend to select them in order, either from the top to bottom or bottom to top. This creates a sequence of events for adjacent checkboxes.
ending with a click on the filing buttons.

- Drop-down Menu

A drop-down menu is the best example of a required sequence of clicks, since you always have to click on a menu before you click on a menu item. In a Web page, the `<SELECT>` element requires a click before it is possible to pick an `<OPTION>` element.

**Child Indices** indicate the structural position of the element in the Web page. When the user is provided an ordered list of objects, the frequency of use of elements will often vary. In Gmail, lower index messages are more frequently read since they arrive chronologically, with the newest messages at the top. The difference in what the user clicks on resulting from child indices is demonstrated in Figure 3.4. With child index information, messages at the top of the inbox are more likely to be clicked on. Without child index information, all messages have a uniform weighting.

![Figure 3.4: Weighted Elements (a) with Child Index Ordering, (b) without Child Index Ordering](image)

**Node Control Names** indicate an explicit label placed on the element by the developer to uniquely identify the element. Anchor and frame elements have names to direct page loads to them. Form elements and their components (e.g. buttons and text fields) have names
to identify their key-value pairs used for posting data. For example, a login form typically has two fields named “username” and “password” so the login button can send the data “username=easylink.project, password=letmein” to login to Gmail. Radio and checkbox elements share these names, so their value is included to identify them. Named elements are better identified by their node control names than by their structural position, since the names are coupled with the underlying tasks.

Text Content was considered because it can identify some equivalences better than structural data. Specifically, the text label of a link may more accurately describe it than its child index.

For example, Gmail chat reorders its Quick Contacts based on availability, although a contact’s name is often more relevant. The erroneous difference in user interaction resulting from repositioned content is demonstrated in Figure 3.5. This figure shows that the second index contact is regularly used. Most of the time Valerie is the second contact, except occasionally when the first contact is not logged in. Not using text content as a feature results in slower training of the quick contacts, but sufficient data will enlarge both the first and second index contacts as they are both clicked on. Child indices is able to model this task, just not as well.

![Figure 3.5: Weighted Elements](image)

(a) (b)

Figure 3.5: Weighted Elements (a) with correct weighting, (b) with erroneous text content
Another possibility is that the user is more interested in a substring of the text content than
the entire string. Links often contain more than one word, and the user may be looking for
specific information. For example, consider a user reading a news feed. A user interested
in Bosnia will open more articles containing the word Bosnia than a user not interested in
Bosnia. Without exhaustive training techniques such as a bag-of-words, which measures
the frequency of each individual word, the use of substrings as a feature is not worthwhile.

Finally, saving any text content across sessions introduces problems with privacy. Since
storing any confidential content such as email details is unacceptable, text content was not
used as a feature.

With all of the features reviewed above, each recorded mouse click generates a concise repre-
sentation of the event. With this representation of events, it is possible to identify equivalent
elements. Examples of recorded mouse clicks are shown in Table 3.1. Each recorded click is saved
as a unique XPath identifying the features of the element that was clicked.

<table>
<thead>
<tr>
<th>Login Username:</th>
</tr>
</thead>
<tbody>
<tr>
<td>/HTML/BODY/TABLE[3]/TBODY/TR/TD[2]/TABLE[1][@class= &quot;form-noindent&quot;]/TBODY/TR/TD/DIV</td>
</tr>
<tr>
<td>/DIV[@class= &quot;body&quot;]/FORM/TABLE/TBODY/TR[3]/TD[2]/INPUT[@name=&quot;Email&quot;][@class= &quot;gaia le val&quot;]]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compose Message:</th>
</tr>
</thead>
<tbody>
<tr>
<td>/HTML/BODY[@class=&quot;sb bn&quot;]/DIV[2]/DIV[1]/DIV[1][@class=&quot;nl&quot;]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sign Out:</th>
</tr>
</thead>
<tbody>
<tr>
<td>/HTML/BODY[@class=&quot;sb bn&quot;]/TABLE/TBODY/TR/TD[3]/TABLE/TBODY/TR/TD[@class=&quot;trb&quot;]/A[2][@class=&quot;sol&quot;]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First Unread Message’s Subject and Body:</th>
</tr>
</thead>
<tbody>
<tr>
<td>/HTML/BODY[@class=&quot;sb bn&quot;]/DIV[2]/DIV[2]/DIV[2][@class=&quot;fs&quot;]/DIV/FORM[@name=&quot;af&quot;]/TABLE[@class=&quot;tlc&quot;]</td>
</tr>
<tr>
<td>/TBODY/TR[1][@class=&quot;ur&quot;]/TD[5]</td>
</tr>
</tbody>
</table>

Table 3.1: Examples of recorded XPaths from Gmail
3.4 Recording

EasyLink works over multiple Firefox sessions by saving to disk when a page is no longer in use and reloading page data when it is later accessed. Page data is stored on disk as a chronological log of recorded events (such as the XPaths in Table 3.1). When a page is first accessed, the user task model expands this chronological log into a complete representation of the user’s tasks, including: a table with a click count of each element, and pairing information for sequential clicks. Additionally, the complete representation contains the last trained event from the current session, which is used to determine when to record new sequential clicks.

3.4.1 Listeners

All model data comes directly from user input. The mouse down and key down events are recorded in the Web browser before any Web page scripts are notified, ensuring that the event’s target has not been altered by page content.

For more complex input such as drag-and-drop and drop-down menus, use of mouse up and key up events are also necessary. If the target of these events is not on the same document, the event is discarded. This prevents page changes on mouse down (such as Gmail messages) from training on whichever element is created at the location of the mouse. Some legitimate sequences may be lost, but recording all sequences of clicks across all Web pages does not scale well. Thus events like dragging a picture from one Web page to another would be discarded.

3.4.2 Content Selection

Every time a training listener fires, the event may provide unnecessarily redundant or misleading information that can result in overtraining and mistraining. To minimize this problem, EasyLink selects which events it will train on.
Repetitive tasks may involve clicking on the same element multiple times, but the amount of information EasyLink gains from subsequent clicks is negligible. While repeatedly clicking, the user is already pointing at the element and does not need to move to click on it. Only the first in a set of redundant events is recorded. The last event time used for sequences of clicks is refreshed on every repeat click.

User error can often lead to mistakenly clicking on an element with no real effect. In order to best prevent training on these errors, clicks within 500 milliseconds are removed from the model. Only the last click in a rapid sequence of clicks is recorded.

3.4.3 Scalability

EasyLink needs to perform all of its calculations in a timely manner, regardless of what the user does in the browser. Therefore, it is worthwhile to evaluate the scalability of its model.

Task models are stored for individual URLs. There is potential for improvement to identify common structures within multiple documents, e.g. a navigation bar shared across pages on one domain. But a naive implementation of this would result in under-emphasis of page content, which is more important than the shared navigation information.

Since Firefox supports many Web pages open at the same time, multiple logs are cached to prevent context switching. As individual logs grow too large, only the more recently recorded events are kept and the older events are discarded. This size constraint prevents overtraining the user task model and maintains the freshness of the model data. If a developer changes their Web site, a smaller model will adapt more quickly. As data are loaded off the disk they are compared against the Web page to determine the element weights. (Recall that element weights reflect the frequency of use). This is the slowest operation in EasyLink. Large Web pages like Gmail take 150 milliseconds to build this data structure.

In order to identify the highest entropy distribution – a uniform distribution across all elements.
in the page – EasyLink needs at least twice as many recorded click events as the number of clickable elements. Otherwise, low frequencies will not be observed. To keep data as fresh as possible, twice the number of clickable elements is a reasonable number of recorded click events.

With respect to the number of clickable elements, persistent data on disk grows linearly and task model operations grow $O(n \log n)$.

### 3.5 Distribution Space

Making trusted changes to the view of a user’s page requires that the user task model have a well specified, understandable interface for its weights. All weights reported by the task model are normalized to the range $[0,1]$ such that the average weight of all elements in the page is 0.5. Elements with weight less than 0.5 are unlikely to be interacted with, and elements with weight greater than 0.5 are likely to be interacted with. As the variance in element frequencies increase with log size, the task model grows more confident in its reported values. So a weight of 0.7 has higher confidence in its likelihood of being interacted with than a weight of 0.6. Weights are generated by Equation 3.1. Element size is included because smaller objects are in more need of enhancement. Weights are returned in a log scale because slow growth of emphasized elements is desired to gradually change the user’s view of a Web page.

Later sections of this thesis refer to important and unimportant elements. Important elements have an above average weight (greater than 0.5). Unimportant elements have a below average weight (less than 0.5).

$$\text{weight} = \frac{\log \left( \frac{\text{clicks(element)}}{\min(\text{element width, height})} + 1 \right)}{2 \log (\text{median clicks/px} + 1)}$$

(3.1)
Chapter 4

Gmail Task Study

The purpose of the long term task analysis is to build a corpus of real data with which we can evaluate user tasks outside of a laboratory setting. Obtaining significant results from a laboratory environment would not mean the model will accurately predict users completing their own tasks outside a laboratory. In order to guarantee external validity, the user task model needs to be able to predict real user activity. Equally importantly, we can use this data to resolve several questions about the task model:

- Is it able to generate a consistent frequency distribution for a given user?

- How much data is sufficient to model a user's tasks?

- How does the model compare within subjects when the data size is varied?

- How does the model compare between subjects?

- What is the maximum duration of time necessary to capture sequences of clicks?

- What is the maximum duration of time necessary to discard misclicks?
4.1 Method

I recruited 6 volunteer subjects who use Gmail daily. Each subject ran a compatible version of Firefox with an extension installed to record data from input events without user involvement. The extension intentionally recorded more information than necessary to reduce the chance that an important feature was missed. Every mouse down, mouse up, and click event on a document in the mail.google.com domain was recorded. The event name (e.g. mouse up, mouse down), time stamp, and XPath (such as the XPaths in Table 3.1) were recorded. Also, the first two characters of text content were recorded to compare labels and child indices without recording confidential information. When the recorded data grew large or the browser closed, the extension flushed these data into a log file in the user’s home directory.

Each user was recorded for at least one month and the resulting data logs were used to train individual user models offline. Different size models were compared to find the smallest model which accurately predicted the user’s behaviour. Models of various sizes from 50 to 700 clicks were trained and tested against the next sequential set of data. Each test element the model did not correctly predict as either likely or unlikely were considered test errors. The percent test error was used to evaluate how accurately it predicted those data. The results of these tests are shown in Figure 4.1.

A second test was run to identify the timing of sequential clicks and misclicks. The user task model was trained only with sequential clicks with different time thresholds. The minimum threshold was determined by evaluating the task model’s accuracy with different time thresholds. The maximum threshold was determined by evaluating what time threshold captured the bulk of sequential clicks.
4.2 Results

Users were consistently modelable. On average, based on user data with a training size of 700 clicks, the model had 36% test error in its trained weights. As the training data size increased, the model grew more accurate.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Log Size</th>
<th>Unique Clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18037</td>
<td>7954</td>
</tr>
<tr>
<td>2</td>
<td>9044</td>
<td>3570</td>
</tr>
<tr>
<td>3</td>
<td>11598</td>
<td>4965</td>
</tr>
<tr>
<td>4</td>
<td>8949</td>
<td>3824</td>
</tr>
<tr>
<td>5</td>
<td>7977</td>
<td>2819</td>
</tr>
<tr>
<td>6</td>
<td>15328</td>
<td>4544</td>
</tr>
</tbody>
</table>

Table 4.1: User Data Size

![Figure 4.1: Training Model Evaluation](image)

4.3 Discussion

From the data, we can conclude that the model was more accurate than random guessing beyond 250 clicks. Achieving beyond 50% accuracy is only possible because of the non-uniform nature of the user's behaviour. Users may be more or less predictable depending on the variety of tasks they choose to complete.

As shown in Figure 4.2, the sequences of clicks within 500 milliseconds provided no information...
Figure 4.2: Click Sequence Minimum Threshold Evaluation

Figure 4.3: Click Sequence Maximum Threshold Evaluation
gain. This suggests that clicks made in a shorter period of time should be discarded because they are more likely noise than directions from the user. Leaving them in would result in a less accurate, slower model of the user’s tasks.

In order to capture the bulk of sequential clicks, we need the upper time threshold to be above the longest duration a user spends moving the mouse between a sequence of targets. Inspecting Figure 4.3, five seconds is a sufficient upper time limit on sequences of clicks. Extending the maximum click sequence threshold beyond that yields minimal gain of captured click sequences and is prone to include clicks that are not sequences.

Different users had similar, but distinct frequency distributions. Even when they were completing the same task, they did so in different ways. Some causes for these differences include how many messages they receive or how much automation they have set up (e.g. the automatic filing of messages into folders). Other factors were what they click on and how they label or delete messages. Examples of different behaviours are shown in Figure 4.4. Some users preferred to click on the sender name for emails; others clicked on the message subject. This may mean users choose whether to open a message dependent on the sender or subject, respectively. Another interpretation is that users keep their mouse closer to the left or right half of the screen.

Figure 4.4: User Data that Clicks on (a) Message Sender Names, (b) Message Subjects
Chapter 5

Browser User Interface

With an accurate model of what the user is likely to interact with, we can dynamically alter visited pages to cater to the user's needs. This chapter discusses the different ways EasyLink changes a Web page's look and feel. These changes are referred to as enhancement modules.

The enhancement modules in EasyLink consist of two visual enhancements (size and contrast) to reduce search time and one motor enhancement (mouse controller) to reduce pointing time. The size module scales up the size of important elements. The contrast module reduces the contrast of unimportant elements. The mouse controller module slows the mouse speed over important elements. Each individual enhancement should benefit either the user's visual search, motor movement, or both. The programming interface that enhancement modules must implement is described in Appendix B.

Every time a Web page changes, all enhancement modules asynchronously process the page. Clickable elements are processed in the order they are likely to be interacted with, starting with the most frequently used. This provides instantaneous feedback about the elements most likely to be interacted with and, due to font resizing, finishes page layout changes as quickly as possible. In order to maintain user agency, not all elements may be processed immediately. Any remaining
elements are rescheduled in the JavaScript event queue.

5.1 Visual Changes

Visual changes have the greatest impact on a user’s opinion of EasyLink. Whether or not a user wants to use EasyLink will likely be determined solely from these changes. These changes are simple and easily understandable so any viewer will understand why they happened: important elements have their font size increased to improve readability and clicking ease; unimportant elements have their contrast decreased to reduce distraction.

5.1.1 Font Size

The size module enlarges element font size based on their probability of interaction. Because page developers often build dependencies into the size of images or tables, changes to the layout should be avoided. I experimented with shrinking unimportant elements and padding the size to retain the original layout but any font size change appears to trigger the layout manager. For this reason, operating on every targetable element is not feasible. Instead, important elements (element weight > 0.5) have their font size increased in a reasonable amount of time, since the number of important elements is small. Additionally, text elements generally have slack to their layout so an increased size doesn’t typically disrupt the entire page layout. A scaled view of Gmail using the size module is shown in Figure 5.1. Unread, recent messages and frequently used buttons have their font size enlarged. The size module operates on element font-size (in pixels) as described in Equation 5.1.

\[
\text{fontSize}' = \text{fontSize}(1.0 + \text{fontScalar}(\text{weight} - 0.5))
\] (5.1)
5.1.2 Contrast

The contrast module adjusts the color of elements to blend unlikely targets into the background. These changes are less disruptive to the view of the page than those produced by the size module because contrast changes blend existing element styles together. Text elements have their color property interpolated between their foreground and background colors. Other elements (such as images, buttons, and drop-down menus) are handled by setting their opacity since blending their foreground and background colors won’t produce a faded effect in the rendered view.

The contrast module operates on each RGB channel of an element’s color as described in Equation 5.2. The percent opacity was operated on as described in Equation 5.3. A scaled view of Gmail using the contrast module is shown in Figure 5.2. Older, read messages have their contrast reduced.
\[
\text{color}^{'} = \text{color} + \text{colorScalar}(0.5 - \text{weight})(\text{backgroundColor} - \text{color}) \tag{5.2}
\]

\[
\text{opacity}^{'} = \text{opacity}(1.0 - \text{opacityScalar}(0.5 - \text{weight})) \tag{5.3}
\]

Originally colors were interpolated as HSV, but often the hue of the foreground or background was undefined. Additionally, interpolating as RGB results in an interpolated saturation and value without requiring conversion to and from HSV. I considered using the contrast module to increase the contrast of likely elements, but there is a limited amount of gain this can produce. For example, the contrast can not be increased when the background is white and the text is black.

Figure 5.2: Contrast Module: Unlikely elements such as bottom messages in the inbox have muted colors
5.2 Mouse Controller

Unlike the other modules, the mouse controller has complex constraints. It runs in real time by intercepting mouse move events and adjusting the mouse’s speed proportional to the weight of the element it is over.

The mouse speed is controlled to add motor space to an element. A highly emphasized element may add up to 16 pixels to the effective width and height of the element. This produces a dramatic improvement for elements with small targets and a negligible improvement for large, easy-to-hit elements.

The mouse controller uses one of the findings from Chapter 4: clicks within a short time period indicate a sequence of clicks to perform one task. No state of the mouse controller is directly observable, so it may dynamically change weights without distracting the user. The result is that complicated tasks don’t create distracters when the user is having the mouse pass over to a target on the other side.

To alter the mouse’s speed, the mouse controller is implemented using the JavaScript event queue and java.awt.Robot, which issues low-level movement commands to the mouse pointer. Pseudocode of the implementation is provided in Table 5.1.

The mouse speed is derived by comparing the last known mouse location against the event location. Focus events and mouse out of window events correspond to mouse movements that were not captured by our listener. Since the mouse location is unknown, the last known mouse data is invalid. Additionally, computationally expensive processes (such as page loads) may slow processing of the event queue. Since position adjustments should not be noticable, out-of-date mouse events are discarded.

Mouse speed is calculated for the X and Y axes independently as a function of element width for each axis and the element weight. New screen positions are calculated and set via
processMouseMove(event)
{
    if (event queue is valid,  
        lastTimeStamp exists,  
        and event.timeStamp - lastTimeStamp < 5 ms )
    {
        offsetPixels = \min(\maxOffset, element.width, element.height)
        offsetPixels *= speedScalar * (element.weight-0.5)
        mouseSpeedX = element.width/(element.width + offsetPixels)
        mouseSpeedY = element.height/(element.height + offsetPixels)
        dx = (event.screenX - lastScreenX) * mouseSpeedX
        dy = (event.screenY - lastScreenY) * mouseSpeedY
        newScreenWidth = lastScreenWidth + dx
        newScreenY = lastScreenY + dy
        invalidateEventQueue
        robot.mouseMove(newScreenWidth, newScreenY)
        queueEvent(validateEventQueue)
    }
    lastScreenWidth = event.screenX
    lastScreenY = event.screenY
    lastTimeStamp = event.timeStamp
}

Table 5.1: Mouse Controller Pseudocode

java.awt.Robot.mouseMove. Since there may be other movements in the JavaScript queue, those
also have to be discarded.

Originally element weight linearly controlled mouse speed, but results from the study suggest
that users do not use the entire width of large elements. Instead, they had to do more work at
a higher index of difficulty to move through the weighted element. As a result, mouse speed was
changed to increase effective width rather than effective speed.
Chapter 6

Look and Feel Study

The purpose of this study is to verify that the modifications made to the browser facilitated faster execution of user's tasks. Typical tasks in Gmail were chosen and adjusted so that the user only had to click on elements, not think or type.

Changes were evaluated both by subject completion time and by qualitative preferences provided by the subjects. Subjects completed tasks with no changes (the control), mouse movement changes, and visual changes. At a minimum, the mouse pointing changes should improve the task completion speed, and visual changes should be preferred by the user.

6.1 Method

Before beginning, each subject was asked background questions about Gmail usage since EasyLink is intended for frequent Web application users (Figure 6.1). All responses from the user were handled through the Web browser. Each subject completed three common tasks in Gmail: reporting an email as spam, replying to an email, and labeling a message.

For each task, subjects completed a warmup followed by three randomly ordered test groups: no changes (the control), visual changes (both size and contrast), and mouse movement changes.
Dear Sir/Madam,

I am a student from MIT collecting data for my master's thesis, titled "Improving Target Acquisition in Web Applications Using Link Prediction". Your participation is voluntary, you may decline to answer any or all questions and at any point you can decline further participation without adverse consequences. You will not be asked for your name or other identifying information to assure the confidentiality and anonymity of this information.

In a fake Gmail environment, you'll go through the motions of three tasks: marking spam, replying, and filing messages. Each task always begins with the warmup, followed by three randomly ordered variations. All tasks should be completed only using the mouse. This study takes 20-40 minutes to complete, depending on how quickly you complete the tasks.

Thank you for participating in this study

How old are you?

Have you used Gmail before?
  ☐ Yes
  ☐ No

How many hours per week do you spend browsing the web?

Thank you very much for your participation.

Let's Begin

Figure 6.1: Look and Feel Study: Introduction
All weights and visual changes were precomputed and stored in the elements on disk to eliminate task model implementation details. Element weights were derived from a composite user task model generated from a normalized union of the Gmail task model data gathered in Chapter 4. Each warmup and test group ran for at least 100 seconds and either 200 seconds or a fixed number of repetitions whichever came first. Spam, reply and label tasks had fixed repetitions of 32, 20, and 23, respectively. These maximum repetition numbers were chosen so that each of the tasks would have a consistent test duration if the subject completed them quickly.

All pages were displayed on a charging Macbook Pro running Mac OS X Tiger with its screen resolution set to 1440x900. Subjects were instructed to use only the provided wired Dell infrared mouse. All tasks took place in a maximized browser window with a centered inner frame document fixed in size to 835x675 with scrolling disabled. An example of the browser’s view is shown in Figure 6.2. All normal affordances were visible to the user, but only a click on the next element in the task resulted in a page change. All pages used the study are available in Appendix C.

Subject’s mouse events (mouse down, mouse up, mouse move, click) were recorded during the tasks and analyzed later. Analysis extracted all valid movements beginning and ending with a mouse down or mouse up. Typically this measured the amount of time from mouse up to the mouse down at the next target. The only exception to this occurred while labeling a message. This task required the user to select an option in a drop-down menu, which could be completed from mouse down on the drop-down menu to a mouse up over the option element. Targets which took more than one click to successfully click on were discarded as errors.

After all test groups were completed for a task, subjects were asked qualitative questions about that task (Figure 6.3). Subjects were asked for their favorite and least favorite of the four variants in the order they completed the tasks (warmup and a permutation of control, mouse, and visual changes). Subjects were also asked if they had completed similar tasks with Gmail.
6.2 Results

Seven subjects volunteered to complete the 45 minute lab study. Subject ages ranged from 19 to 50 with a median age of 23. The number of hours subjects reported having spent browsing the Web ranged from 10 to 50 with a median of 15 hours. Six of the seven had used Gmail previously. The total data size (number of completed repetitions) for each task and group is listed in Table 6.2. The percent of data discarded due to misclicks is listed in Table 6.2.

<table>
<thead>
<tr>
<th></th>
<th>Warmup</th>
<th>Control</th>
<th>Mouse</th>
<th>Visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spam</td>
<td>180</td>
<td>201</td>
<td>213</td>
<td>208</td>
</tr>
<tr>
<td>Reply</td>
<td>125</td>
<td>132</td>
<td>132</td>
<td>131</td>
</tr>
<tr>
<td>Label</td>
<td>144</td>
<td>151</td>
<td>148</td>
<td>152</td>
</tr>
</tbody>
</table>

Table 6.1: Task and Variant Data Size
Task Evaluation

In the order they were shown to you, what was your favorite variation:

- Don't Remember
- No Preference
- 1st
- 2nd
- 3rd
- 4th

In the order they were shown to you, what was your least favorite variation:

- Don't Remember
- No Preference
- 1st
- 2nd
- 3rd
- 4th

Had you done this task previously in Gmail?

- Yes
- No

Let's Continue

Figure 6.3: Look and Feel Study: Task Exit Questions

<table>
<thead>
<tr>
<th></th>
<th>Warmup</th>
<th>Control</th>
<th>Mouse</th>
<th>Visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spam</td>
<td>3.9%</td>
<td>2.8%</td>
<td>2.8%</td>
<td>5.3%</td>
</tr>
<tr>
<td>Reply</td>
<td>6.2%</td>
<td>6.5%</td>
<td>7.7%</td>
<td>6.9%</td>
</tr>
<tr>
<td>Label</td>
<td>9.3%</td>
<td>12.3%</td>
<td>7.6%</td>
<td>8.0%</td>
</tr>
</tbody>
</table>

Table 6.2: Task and Variant Error Rates

The results of the spam tasks are shown in Figure 6.4. Each of the three targets (the unread message, report spam button, and sign out button) were evaluated for each of the four test groups. Subjects performed the motor variant of the spam task significantly faster than the control ($p < 0.05$). Four of seven users claimed to have reported messages as spam previously.

The tests for effects of EasyLink on both the reply and label tasks were inconclusive, due to less data. Also, users became faster and more error prone at their tasks over time, which further reduced the difference between test groups. The results of the reply and label tasks are shown in
Figures 6.5 and 6.6. Two of seven users claimed to have replied to a message previously. Three of seven users claimed to have labeled a message. These numbers seem unexpectedly low and will be discussed in Section 6.3.

Each task was also evaluated based on the user satisfaction. Of all the test groups, the opinions regarding visual changes were most contentious as they were both the most preferred and least preferred changes. Reported preferences for tasks are shown in Figure 6.7.
6.3 Discussion

The results from the spam task are most promising. The time to click on each target was significantly improved by the mouse movement changes. Additionally, visual changes insignificantly improved the time. The spam task was the first and by far the simplest task provided to the users. This caused several benefits including more data points and less errors caused by the subjects. Short of a long term study with different types of users, these findings are quite encouraging.

Most subjects did not report having replied to messages. Since email is normally a two-way communication, replying to messages should have been the most common task and suggests some aspect to this task artificial. Subjects may have interpreted the question to mean the entire task,
which included the use of Gmail’s labels, or the manner in which the task was completed, which required clicking on the text area rather than the reply button.

The third target and fourth targets of the reply task did not produce expected results. The third target was a large text area of size 483x60, which is relatively large compared to other targets whose heights were never greater than 24 pixels. The mouse variant caused users to spend more effort within the text area than the control. This resulted in higher movement times to this target. This problem has since been addressed by adjusting the design of the mouse system to increase motor space by effective width rather than by mouse speed. The fourth target was simply a poor choice for a pointing task. Since the third target surrounds the fourth target, the movement time depended more on the cursor’s start position than time to move to the fourth target. This is the most likely cause of the high variance for this target.

The results for the label task were unfortunately biased due to a bug in the mouse controller at the time. The drop-down menu sometimes caused the mouse cursor to jump to the upper left, depending on the ordering of events in the JavaScript queue. This may have influenced users to report the mouse variant of this task as the least preferred, even though they were instructed that the bug was not intended as part of the study. Both mouse controller issues have been resolved since the study was completed.

After the study finished, some users expressed their concern that the visual scaling felt artificial for their task. In fact, since element weights were derived from actual data, weights were not artificial, but also there were distracter elements that went unnoticed by the subjects. This feedback may explain the large qualitative contention over the visual changes. Regardless, a user should be able to configure their view of EasyLink, so users who do not like visual changes could disable them.

The implications regarding the effect of EasyLink may depend on the type of task. The
benefits for various audiences doing more complex tasks should be explored with more data. It may have been possible to evaluate this by randomly ordering the tasks and grouping all variant work together, but this change certainly would have increased the variance in the spam task’s results with other confounding variables. Qualitative evaluations would also receive one third its data if this were the case.
Chapter 7

Conclusion

This thesis has presented EasyLink, an extension to facilitate expected user interaction in Web applications. EasyLink’s design and performance oriented implementation enable it to support user tasks in Web applications. The user task model is robust to the user’s behaviour. Visual changes enhance the view of the page as it loads. And the mouse movement makes it easier to accomplish frequent tasks. This chapter concludes the thesis with an overview of the academic contributions, and work that should be done to further EasyLink’s validity and usability.

7.1 Contributions

There are several contributions this thesis has presented. Since subjects were able to adapt to the dynamic changes in the mouse controller, procedurally generating motor space changes are not too confusing to users. EasyLink automated the control display gain using task model weights. Additionally, the X and Y axes of CD gain were set independently, so there could be a high gain in one axis and low gain in another.

EasyLink demonstrates that applications are ready for more advanced alterations to the look and feel of pages based on predictive behaviour. Study results suggest that improving target
acquisition benefits the user when key elements are emphasized even with distracter targets.

EasyLink records the user task model with the XPaths of elements clicked on, not the URLs associated with them. This provides a more robust mechanism to identify what parts of a Web page are used rather than what Web pages are used next.

EasyLink provides new techniques for transforming Web pages visual and motor space in real time. Operating on the elements as groups and prioritizing modifications gave immediate feedback to the user.

7.2 Further Work

EasyLink is not a complete, polished extension. There are several important aspects missing in its current form, but combined results from both studies are encouraging. Both expected features and further improvements are missing from its implementation and a breadth of testing on other applications is needed to validate its usability and effectiveness.

7.2.1 Full Evaluation

The evaluation presented in this thesis is preliminary. Some results of the user study were significant, but many of the tests were inconclusive or even suggested problems with EasyLink’s implementation. The problems have been addressed with design changes or bug fixes, but more testing in a non-laboratory setting is recommended.

One way to gather field data is to test volunteers using the extension for a long period of time on their own computer. Choosing a set of common sequences of clicks would enable us to test the user by randomly assisting with those tasks and measuring the resulting time. Instances where we do not assist the user would act as the control group, and instances where we assist the user would act as the experimental group. Since we know users perform certain tasks in field settings,
we can evaluate these tasks without compromising the validity of the study.

7.2.2 Additional Features

EasyLink has been developed and tested with a month of data for a handful of users. For EasyLink’s usability to be sufficient for a larger user base over a longer period of time, EasyLink needs additional features. EasyLink needs to provide a user interface to its settings, to handle visual information, to address security issues, to discard unused Web page data, and to further improve its performance.

Configuration User Interface

Currently there is neither a means to adjust the severity of alteration of documents nor a means to disable some of the enhancement modules. This user interface does not need to be complex, but users want high-level control of the extension. A simple preferences dialog would suffice.

Visual Information

Some content on pages is not intended to be clicked on (such as images and paragraphs of text). Text is handled by scaling up the weight as though it were one line of text, but some unwanted text may be more than one line, such as Google advertisements. Since lower contrast images did not seem as bothersome, images have not been addressed at all. Both of these are minimal work on visual data and they should be further iterated on.

Keyed Security

In order to support sensitive information concerns, stronger security measures should be incorporated, as well as a stronger evaluation of the overall security.
Long-Term Garbage Collection

A long-term implementation of EasyLink needs to address stale content in its records. General Web browsing creates many cached files with negligible clicks. EasyLink should evaluate the longevity of stored data with consideration for the amount of recorded data.

Asynchronous Cache Loading

The largest performance bottleneck in EasyLink is the fetching and preprocessing data from disk into a usable form. Large data sets like Gmail may take up to 150 milliseconds to prepare. EasyLink should load data asynchronously and block the enhancement modules until the data is ready. In order to keep EasyLink responsive, it may need further development of its stored data structure to permit likely targets to be processed immediately.

7.2.3 Additional Web Application Evaluations

Gmail has been the primary testbed for this project as my primary mail client, but a broader spectrum of applications should be evaluated by people who regularly use them. For example, EasyLink has given high weights to hard-to-click elements in Google Calendar (Figure 7.1). This suggests EasyLink would improve the index of difficulty of adjusting scheduled event lengths. No thorough testing has been done, so an evaluation of other Web applications such as Google Calendar is recommended.
Figure 7.1: Cursory Google Calendar Data. The end interacters of scheduled events have high weights.
Appendix A

Task Model Interface

Element[] modelGetElementChunk(node);

Returns the group of Elements to which the Node belongs with the following properties:
the returned array has non-zero length; parent Elements are always earlier in the array than
child Elements; if the Node is an Element, it will be within the array, otherwise its parent
Node will be within the array.

Requires Node != null.

Weight modelGetAffordanceWeight(node);

Returns the Weight information for the specified node, which contains the following information:

nodeWeight - The node’s current weight irrespective of the previously trained event.

pairedWeight - The node’s current weight only considering unpaired data and the previ-
ous target’s paired weight.
void modelDocumentChanged(rootChangedElement);

Notifies the model of all changes to the document. Ensures the document’s model data is
loaded and calculates global statistics on usage.

void modelTrainEvent(event);

Notifies the model of an event which it may train.

void modelCloseAllData();

Notifies the model that it should flush all its data to disk. Used to close all logs on exit.
Appendix B

Enhancement Module Interface

void scaleElement(element, weight);

Provides an Element and its associated weight for the module to operate on. The module should not operate on other elements.

void initializeModule();

Initialization hook for the module. Used to construct the mouse controller’s Robot and set up mouse move listeners not available through EasyLink’s standard interface.
Appendix C

UI Study Pages

Spam Task Description
You’ve received spam, and want to get rid of it properly
Steps:
1. Sign in
2. Open the unread message from Roger Hanna
3. Report this message as spam
4. Sign out

Start the Task

Figure C.1: UI Study Spam Description

Reply Task Description
You’re responding to a message sent to reuse to claim some goods.
Steps:
1. Sign in
2. Open the reuse label
3. Open the unread message from Roger Hanna
4. Click in the text area to begin a reply to this message
5. Send
6. Sign out

Start the Task

Figure C.2: UI Study Reply Description
Label Task Description

You've received a bcc'd message and you want to file it appropriately.

Steps:

1. Sign in
2. Open up the unread message from Roger Hanna
3. Click on the "More actions..." dropdown to get a list of labels
4. Apply the label random-hall to this message
5. Archive this message
6. Sign out

Start the Task

Figure C.3: UI Study Label Description

Figure C.4: Study Spam Page 1: The user clicks on the first, unread message in their inbox.
Figure C.5: Study Spam Page 2: The user clicks on the report spam button.

Figure C.6: Study Spam Page 3: The user clicks on the sign out link.
Figure C.7: Study Reply Page 1: The user clicks on the label with an unread message.

Figure C.8: Study Reply Page 2: The user clicks on the first, unread message in their inbox.
Figure C.9: Study Reply Page 3: The user clicks on the text area to begin a reply.

Figure C.10: Study Reply Page 4: The user clicks on the send button.
<Figure C.11: Study Reply Page 5: The user clicks on the sign out link.

Figure C.12: Study Label Page 1: The user clicks on the first, unread message in their inbox.>
Dance Troupe Shows Next Weekend

Roger Harris

Dance Troupe Shows Next Weekend

Roger Harris

Don't miss

GyroHazard

Dance Troupe Concert

Carnival Contents May Be Wet!

Little Kresge Theater

Thursday, 12/7 - 8pm $5

Don't miss

GyroHazard

Dance Troupe Concert

Carnival Contents May Be Wet!

Little Kresge Theater

Thursday, 12/7 - 8pm $5

Don't miss

GyroHazard

Dance Troupe Concert

Carnival Contents May Be Wet!

Little Kresge Theater

Thursday, 12/7 - 8pm $5

Figure C.13: Study Label Page 2: The user clicks on drop-down menu and selects the random-hall label.

Figure C.14: Study Label Page 3: The user clicks on the archive button.
Figure C.15: Study Label Page 4: The user clicks on the sign out link.
Bibliography


