iAnnotate: Exploring Multi-User Ink Annotation in Web Browsers

Beryl Plimmer, Samuel Hsiao-Heng Chang, Meghavi Doshi,
Laura Laycock, Nilanthi Seneviratne
Department of Computer Science
University of Auckland
beryl@cs.auckland.ac.nz, {hcha155| mdos002|llay008| lsen008}@aucklanduni.ac.nz

Abstract

We present iAnnotate, a tool that provides multi-user digital ink annotation on standard web pages within a commercial browser. The annotation can be saved, retrieved and shared with others via a URL. In addition multiple users' annotations can be displayed on the same page. We describe our design goals and the technical challenges. While realizing annotation on web documents is difficult because of the dynamic nature of the documents and the security constraints of web browsers, our user evaluation suggests that fully realized digital ink annotation tools would be very valuable.

Keywords: Digital ink, document annotation, web annotation

1 Introduction

Annotating a document with a pen helps people think and engage more deeply with the material (Wolfe 2000). Furthermore annotations can be used to communicate ideas about the document content with others (Marshall 1997). Increasingly documents are web-based. Yet, although the computer hardware to annotate documents (digital pen and touch interfaces) is freely available, there are few examples of ink annotation functionality in web browsers.

Digital ink annotation support is available in many desktop applications such as word processors. However web browsers do not offer built-in annotation functionality of any kind. In terms of retrieving and sharing information, web browsers are probably used more than desktop applications. The existence of annotation functionality, enhanced with storing and sharing, should further increase the convenience offered by the web.

Web documents and web browsers present particular challenges that make annotation difficult. First, web documents are usually dynamic. That is the content is changing quickly and web documents are designed to reflow within the available window space. Annotations derive meaning from their location, therefore robust location fixing of the annotation within the dynamic document is required. Second, the web browser is on the user's machine and from this machine it interacts with data from unsecure sources. As such browsers present a security risk and must have excellent security. The security of browsers limits the extension points for add-ins and the ability to mix content between different servers. Finally, realizing digital ink annotation is technically quite difficult as text, pictures and annotations must lie over each other. This usually requires layering of the interface which can be problematic as accurate position information of elements on the different layers is not always available.

Several projects have investigated user annotation of web pages (e.g. Cadiz, Gupta et al. 2000; Chatti, Sodhi et al. 2006). However, none of these applications provide a complete solution. Some lack the flexibility to allow annotations to adapt to the changes of the underlying webpage; others do not save the ink or lack support for window scrolling.

Digital ink, created with a stylus directly onto the display, simulates real world annotation behaviour by enabling users to directly draw on web pages. The interaction is straightforward. However a number of complex problems are faced from an implementation perspective. In this project we investigate whether Silverlight (Microsoft Corporation 2008), a recently released browser plug-in with digital inking capabilities, provides sufficient functionality to support freehand annotation on any web page.

2 Related Work

Current implementations of annotation on the web can be divided into two types: text based annotation applications (Wilson ; Kahan, Koivunen et al. 2002; Bottoni, Civica et al. 2004) and ink based annotation applications (Ramachandran and Kashi 2003; Chatti, Sodhi et al. 2006). Text based annotation applications allow the user to annotate web pages using the mouse and keyboard whereas ink based annotation applications allow the user to utilize a stylus or touch screen (or mouse) to annotate a webpage with digital ink. Both approaches need to be able to capture and display the annotations, store and retrieve annotations, and finally allow for changes in the underlying webpage and size of the browser window.

An important aspect of annotation is deciding how to acquire the annotation from the user. In text based annotation systems, this entails the user typing the annotation using a keyboard. In digital ink annotation the user is able to annotate directly on top of the document, as if it were paper.

Text based annotation applications generally require the user to select text, right click the selected text or click a button to generate a text field to enter the annotation (Figure 1). This can be unintuitive and also breaks the thought process behind the annotation. This means that

the user's concentration is on how to insert the annotation rather than the actual content of the annotation.

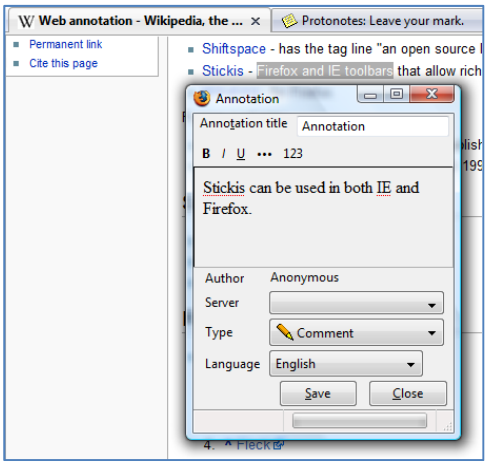

Figure 1: Annotating with Annozilla (Wilson)

Digital ink annotation bypasses this aspect of annotating because the user is able to directly write on the webpage. This way the user can concentrate on the content of the annotation. Although this process can be achieved via a mouse it is most effective when using a stylus for input.

Displaying the annotations to the user is as important as acquisition of the annotation. In a text based annotation application this usually is achieved by displaying an icon for the annotation –the user double clicks the icon in order to view the annotation. Annotation applications that support digital ink can display the annotation more easily directly on top of the webpage. For example, uAnnotate (Chatti, Sodhi et al. 2006) uses a transparent Flash object so the user is able to view the created annotation as well as the content of the underlying web page.

However, in both cases it is possible to cover the content of the document with annotations. This becomes a problem in digital ink annotations, because the ink is displayed on top of the content. Therefore functionality to hide or filter annotations is required. Annotea (Kahan, Koivunen et al. 2002) allows the user to filter and hide annotations according to author, server or annotation type. uAnnotate (Chatti, Sodhi et al. 2006), provides a link which the user can click to clear all annotations on the web page.

Annotations derive meaning from their position so it is important to retain their relative position when the underlying document changes. Moving the ink to retain its meaning is called ink annotation reflow. Brush et al. (2001) in a study regarding the repositioning of annotations concluded that users prefer the annotation to be anchored onto specific key words. For example if the document is changed and only a small amount of the annotated text is remaining, the user expects the remaining text to be still annotated.

uAnnotate (Chatti, Sodhi et al. 2006) approaches the problem of repositioning when scrolling by making the ink overlay as large as the webpage. This means that when the user creates an annotation and scrolls down the page the annotation scrolls with the content. If the user resizes the window then the overlay as well as the content resizes to fit the window.

When the underlying web page is changed via insertion or deletion, the annotation application should recognize this and display accordingly. In the Avaya prototype (Ramachandran and Kashi 2003) and uAnnotate (Chatti, Sodhi et al. 2006) if the underlying content is modified, the annotation is not displayed. This is acceptable if the keywords for anchoring the annotation have been deleted. However if there is an insertion before the keywords the annotation application should still display the annotation. Annotea (Kahan, Koivunen et al. 2002) does not display orphaned annotations (annotations where the keywords have been deleted) on the web page however they will display it in a list view where all annotations associated with the page are displayed. Current approaches provide approximate reflow in common situations. The dynamic nature of web documents and browsers makes robust annotation reflow a difficult problem.

The transient nature of web pages means that the annotation needs to be anchored properly in order to allow annotation reflow. In order to generate the anchoring points Wang and Raghupathy (2007) describe methods in which the anchoring position can be determined by the context of the annotation. This means that the annotation anchoring position should be determined by the type, as well as the position of the annotation. Another approach to attaching annotations to the underlying document is described by Priest and Plimmer (2006). This method uses a linker, which is the first stroke of the annotation. The linker stroke can either be a circle stroke or a line stroke identifying the document position to which the annotation is attached.

In order to create an effective web annotation application, annotations need to be encapsulated and stored efficiently so that they may be reused at a later date. Annotations are usually encapsulated either using the resource description framework (RDF) or extensible mark-up language (XML) schema. The common annotation framework (Bargeron and Moscovich 2003) and the Avaya prototype (Ramachandran and Kashi 2003) both use a customized XML schema to encapsulate attributes of annotation such as the anchors for the annotation, the annotating document and the annotation content. Annotea (Kahan, Koivunen et al. 2002) uses the RDF framework to create classes of annotations that can give fine grained detail of annotation types. In either case, the annotation can be formed so that it is self-contained. This means that the annotation can be stored on a different server than the web pages. Thus it promotes portability of annotations, and also provides an approach to annotate read-only documents (Bargeron and Moscovich 2003).

Most web annotation applications prefer to encapsulate annotations via XML and save these files in servers (Kahan, Koivunen et al. 2002; Ramachandran and Kashi 2003; Bottoni, Civica et al. 2004; Chatti, Sodhi et al. 2006). There are number of advantages in storing annotations in this manner: all annotations are hosted in one place thus preventing users from loading annotations to the wrong web page. Annotea (Kahan, Koivunen et al. 2002) and MADCOW (Bottoni, Civica et al. 2004) take advantage of this, incorporating multiple servers in their architecture. However, a drawback of this approach is

that each server uses an API to retrieve and store annotations. This means third parties cannot extend the implementation unless they are provided with this API.

As an alternative Denoue and Vignollet (2001) propose that annotations should be stored within the URL of the webpage. As well as being self-contained, it avoids the use of a server altogether. However this approach to storing would make ink annotation storage difficult due to browser limitations such as the character limit for the URL. Also, it does mean that users are unable to annotate on third party web pages due to restricted domain access.

Another option is to store the annotations locally. uAnnotate (Chatti, Sodhi et al. 2006) uses Flash's Local Shared Objects to store information about its annotations. The main drawback of this approach is that annotations are restricted to one computer and one user.

If the user wishes to view annotations which were saved at a previous date, he or she could import this annotation onto the same webpage. For example in uAnnotate (Chatti, Sodhi et al. 2006) it is possible to export the created annotations into an XML file and reload this file to a web page at a later date. In this case the annotation application should check whether the URL of the annotating document and the URL of the web page are the same. If the annotation is saved onto a server, this server should prevent the user from loading an annotation which was not created for that particular web page.

Some of the constraints of existing digital ink annotation are that the annotation pane is restricted to the current window (Chatti, Sodhi et al. 2006), the annotation can be loaded onto any URL (Chatti, Sodhi et al. 2006), and saving is local only or manual from the user's clipboard (Chatti, Sodhi et al. 2006). Reflow is either not supported (Chatti, Sodhi et al. 2006) or limited (Ramachandran and Kashi 2003). Annozilla (Wilson) supports multiple text annotations but only one can be viewed at a time: we are not aware of any digital ink annotation tools that can show multiple users' annotations at the same time.

3 Design Goals

Our vision for this project is a web space where a document can be annotated with digital ink by any number of people. The annotations may have different purposes: for example they may be a teacher's or student's comments on course notes, a study group's shared considerations of a text, or an extended family's shared annotations on a family calendar. Potentially, they could contribute to social discourse where an initial document acts as the trigger for a discussion. Realizing this vision requires us to explore a number of technical and design issues.

Many-to-many annotations raise several research questions: how can we appropriately visualize, index, store, search and filter the annotation data? Imagine a web document that 10, 20, or 100 people have annotated. Simply displaying the annotations over the document is unlikely to be useful or usable. However seeing 'hot spots' (or 'aggregated' annotations), points in the document that a number of people have annotated, is likely to be useful to find foci of attention. Additionally filtering the annotations to see selected users' (e.g. the boss/teacher) or groups' (friends/ colleagues) annotations or those on a particular theme or topic are also likely be useful. If the annotations are a conduit for social discussion then being able to retrieve the annotations with appropriate spatial and temporal information is also crucial. One can imagine converting annotations into a time series so that a viewer can replay the conversation (or the time series could simply be used as a mechanism to see individual annotations in a hotspot).

In this first stage of the project we explore the technical issues of realizing multiple, shareable digital ink annotations on any web page (not just specially designed pages that incorporate annotation) and basic usability considerations for digital ink web annotation.

4 Our Approach

As there are well-documented limitations to current approaches such as Flash, in this project we have investigated Silverlight. Microsoft Silverlight (2008) is a cross browser and cross platform plug-in which allows developers to create rich internet applications. Of particular interest for this project is that it exposes an API to the .NET Framework that has excellent support for digital ink.

From a software architecture perspective there are two parts to realizing the annotation capability. The first is the core annotation framework, which provides 'on document' functionality of inking, anchoring and reflowing ink, and general ink editing support. The second is the browser extension for the instantiation of the ink overlay, and saving and loading of ink.

4.1 First Prototype

Our first prototype (Figure 2) used Silverlight 2 beta 2, released in June 2008. The core annotation framework provides functionality for the editing, anchoring and reflow of the digital ink. The browser plug-in provides a toolbar with start and stop annotating buttons and functionality to inject Silverlight into a page's HTML.

4.1.1 Core Annotation Framework

The core annotation framework (CAF) of iAnnotate was built using Silverlight, JavaScript, XML and ASMX web services. In order to capture the annotation the CAF uses a Silverlight object. When the page is loaded this object is overlaid on the webpage. The Silverlight object is transparent, except the buttons, which allow the user to interact with the application.

The Silverlight object contains our InkPresenter control, which renders digital ink strokes using the movement of the stylus. The InkPresenter can also create strokes programmatically thus it is used to display previously saved annotations. When the user begins an annotation the Silverlight object calls JavaScript methods to anchor the annotation to the underlying webpage.

To anchor annotations effectively the individual strokes must be grouped into annotations. Two approaches were considered. The first serializes the ink strokes into a XML string and sends this to the server where they can be recognized and grouped into different annotations. However, this method may take a considerable amount of time as the strokes need to make a round-trip to the server.

Figure 2: Prototype one

The second method uses a bounding box to group ink strokes into annotations. The bounding box implemented is similar to that used by Priest and Plimmer (2006).

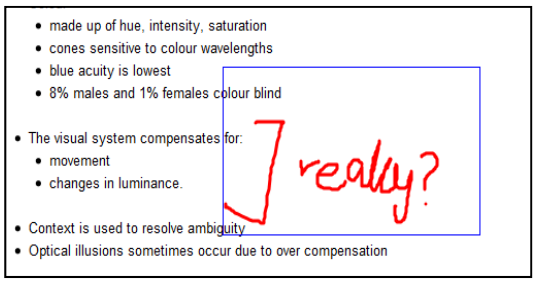

Figure 3: Bounding box in CAF

This bounding box is first displayed to the user when they start an annotation. As seen in Figure 3 all strokes (red ink) within the bounding box (blue rectangle) are classified as one annotation. If the user starts an ink stroke within the bounding box and reaches towards any edge of the bounding box the bounding box will expand in that direction to allow the user to carry on annotating.

The algorithm for grouping ink strokes checks if the first point of a new stroke is inside the bounding box of the current annotation. If it is the stroke is added to that annotation. If not, a new Annotation object is created and a bounding box created to enclose it. This algorithm can potentially group two separate annotations together. For example if a user circles one word in a sentence and tries to underline another word which is just within the bounding box, then the bounding box would expand to allow this stroke to be in the same annotation.

Once the ink strokes are grouped into annotations, anchor locations on the underlying webpage are computed. The position of the first point of the first stroke is used to identify the closest HTML element. The standard JavaScript method 'elementFromPoint' returns an HTML element given a point. However it only returns the first element enclosing the point. Thus when the CAF is on a webpage it always returns the Silverlight object and not the content elements.

We developed a JavaScript algorithm to determine the closest HTML element. This algorithm traverses through the Document Object Model (DOM) tree to find the vertically closest HTML element in the document's body section. The algorithm matches the vertical position of the annotation to the document because the distance between

the annotation and the top edge of the document describes the section that the annotation belongs to most accurately.

The main limitation of this algorithm is that if there are a large number of elements on the webpage it will iterate through all of the elements. This may delay the rendering of the first stroke. Another constraint is that the annotation is anchored to an element; JavaScript does not allow fine-grained control over the text in a webpage. However, attaching the annotation to the element has the advantage of supporting anchoring annotations to images as well as text elements.

Annotation reflow happens on two occasions: when the browser window resizes and when the underlying content of the webpage is changed. The algorithm processes each annotation by first locating its anchoring element. If it is an element other than the document element it computes the x and y offset of the anchoring element from the top of the page. This offset is used to move the annotation. In the case when the anchoring element is the document element of the webpage the ink annotation is rendered at the original position. If the anchoring element is not found the annotation is an orphan and is not displayed.

Our reflow algorithm does not properly take account of the horizontal position of an annotation. If the browser window is resized then the annotations can potentially be on the wrong place. More sophisticated reflow is necessary for a completely satisfactory system.

4.1.2 Plug-in

Our goal with the plug in is to extend a browser with a toolbar to provide the necessary interaction buttons (‘Start Annotating’ etc.). Silverlight works best with Microsoft Internet Explorer (IE), so as a proof of concept we concentrated our efforts on IE. iAnnotate’s plug-in separates into two main sections: the Browser Helper Object, and the iAnnotate toolbar

An Internet Explorer extension can be created by using the Browser Helper Object (BHO). BHO allows the developers to access Component Object Model (COM) components that load each time the browser starts up. The BHO implements the IObjectWithSite interface to establish a COM-based communication channel to obtain the browser’s events.

The iAnnotate toolbar of the extension inherits the BHO. The BHO adds the band on which the buttons can be placed. iAnnotate adds the “Start Annotating” and “Stop Annotating” button in the band of the toolbar displayed on the browser, as shown in Figure 2.

On ‘start annotating’ the plug-in injects a Silverlight object into the webpage. Our first attempt injected the Silverlight object into the browsing pane of the browser: the area in which the webpage is displayed. While we could add the Silverlight object, the events were not fully exposed. Thus we could not progress with this approach.

The second approach injected a Silverlight object into every webpage the user wishes to annotate. To achieve this, the Microsoft HTML Object Library was used. This “mshtml” reference allowed access to the webpage’s HTML through the Component Object Model (COM). This access was obtained by creating an HTML document using the HTMLDocument class and casting the Browser’s document into the HTMLDocument type. When injecting HTML using the mshtml “insertAdjacentHTML(string where, string HTML)” method, the injection of a reference to an external JavaScript file does not work. We attributed this to security. However injection of inline JavaScript was effective.

We converted all of the JavaScript code into one string and added it using the “execScript(string code, string language)” method. Thus a clean architecture could be obtained and the extension could successfully inject the Silverlight object into any webpage the user would wish to annotate. There were additional problems with interfacing between the browser and server for saving and retrieving annotations that we had not fully resolved before Silverlight 2.0 was released.

Using this version of iAnnotate we undertook an informal usability study, which is reported in the evaluation section.

4.2 Second Prototype

The upgrade to Silverlight 2.0 (release version) in November 2008 caused significant problems with our plug-in architecture. Unfortunately the release version of Silverlight 2 disallowed applications to be retrieved from outside the site’s domain. As a result, although we could still insert code into web pages and call Silverlight from the page, because the domain of that page is different from the annotation server it will not respond.

After exploring various avenues we decided the only viable approach was an IFrame design. Similar to the previous approach, we have layers. The top layer is the annotation panel made with Silverlight 2.0, which allows the creation of digital ink. The middle layer is the IFrame, which displays the content of some webpage ready to be annotated. These layers sit on a HTML page, which is located server side, which means the Silverlight can be loaded without security issues.

Annotations can be saved and retrieved from a simple MySQL database on the server. Each annotation is tagged with the user’s ID, URL and time data. This allows annotations to be shared and multiple annotations of the same page displayed.

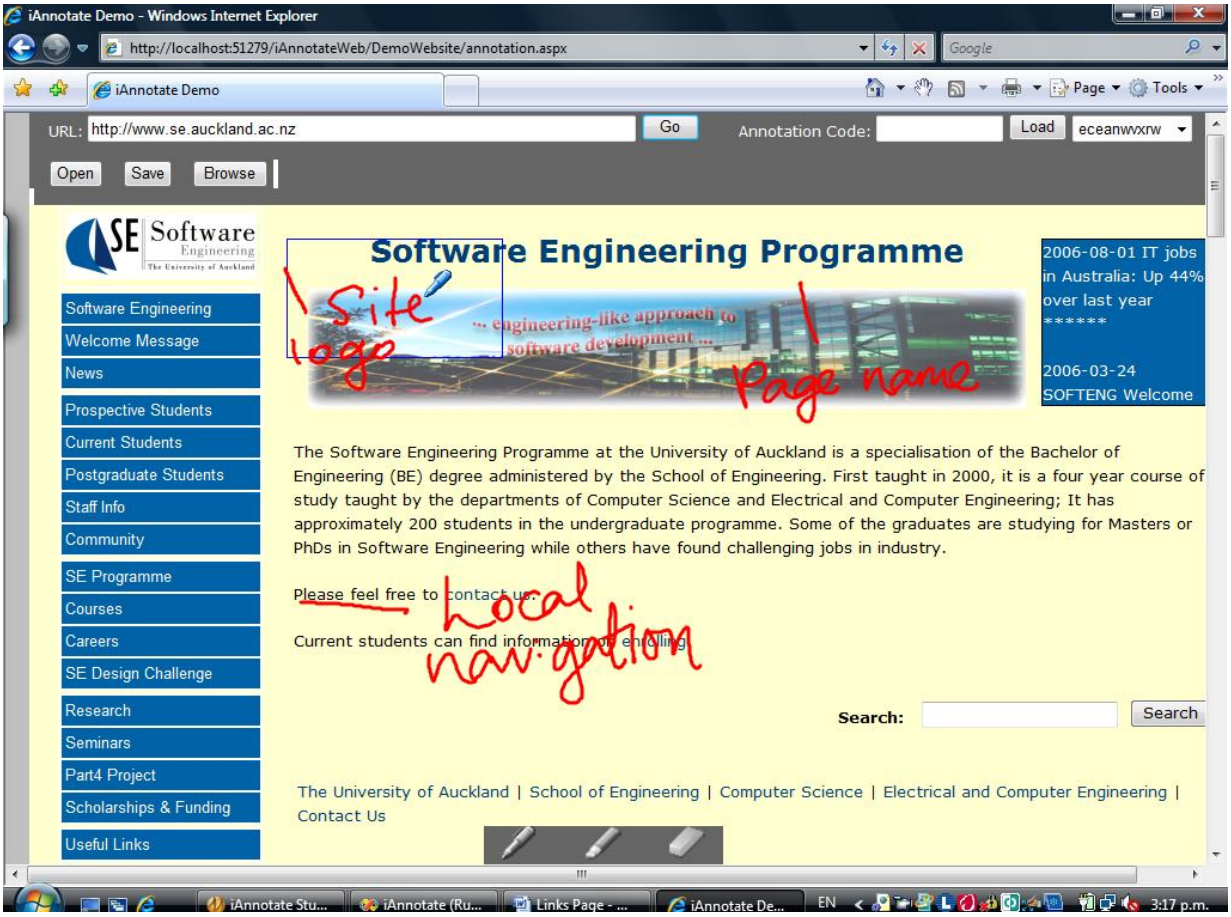

Figure 4: Prototype two

The user experience with this implementation is different. The user must login to the annotation website so that their annotations can be identified. In order to annotate a page the user must first put the URL in the textbox at the top of the page and click ‘go’ to load the page (Figure 4). They can then ink, highlight and erase using the buttons at the bottom of the window. On saving each annotation is automatically allocated a unique key and a URL is generated for the key (annotation URL with the key appended) (Figure 5). The annotation can be retrieved using the key or the URL. Using this generated URL the web page and annotation can be retrieved without logging in. Multiple annotations for the one page can be loaded by entering the key or selecting the annotation codes from the list and clicking the ‘load’ button (Figure 6). Annotations of a different page can be loaded with the page from the open list (Figure 7). This is a two-step process; first the page is loaded from its original location (we do not copy the page content to our server) then the annotation is rendered on the page using the CAF.

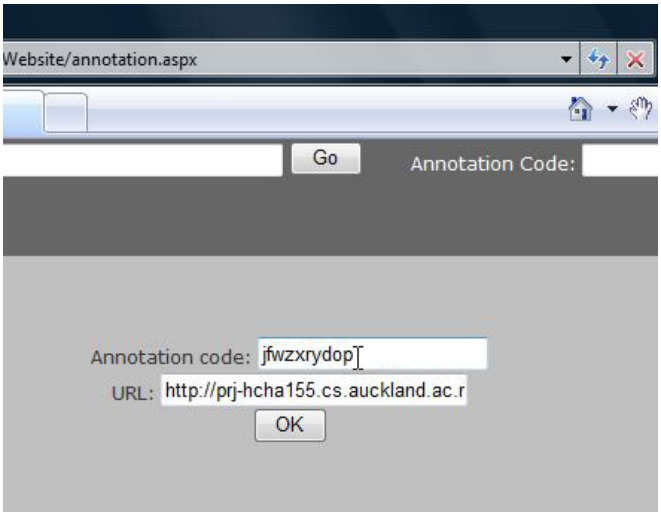

Figure 5: Save URL

Figure 6: Load list

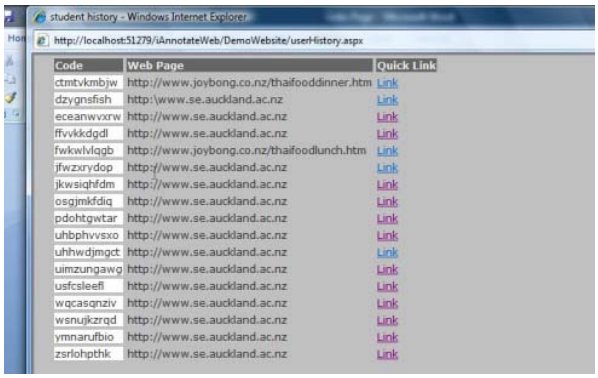

Figure 7: Open list

The disadvantage of this approach is that the user has to copy the URL of a page they want to annotate into the field on the annotation website. The annotation website then reloads the webpage into the IFrame. This is a less natural interaction paradigm than we would like. Furthermore it is difficult to calculate the size of a webpage before it is loaded consequently we have created a fixed (large) IFrame that can accommodate most standard web pages (there are some dynamic technologies such as AJAX that do not work). This effectively disables normal window resizing functionality as the IFrame is of fixed width. One benefit from fixing the width is that ink reflow is much less of an issue. Another advantage of this approach is that the user does not need to install a plug-in so the functionality is more freely available.

5 Evaluation

We have conducted user evaluations on both versions of iAnnotate. The usability study on the first prototype evaluated three main aspects of the system: the user interface, the appeal of digital ink annotation in a web browser and the users' reactions to annotation reflow. The usability of the second prototype focused on the sharing of an annotation.

5.1 Prototype One

The study was conducted with eight student participants on a Tablet PC with stylus input. The participants had used a stylus at least once and were familiar with how a Tablet PC works. Most of the participants digitally annotated paper documents at least "sometimes" but "not often".

The study consisted of two main parts. The first part was to check the usability of creating annotations. The participants were provided with a paragraph in a wiki which had several spelling mistakes. They were asked to mark-up the corrections with ink annotations (Figure 8 top). Once this task was accomplished, the participants were asked to rate the system for its user interaction and functionality.

The second part was to examine the ink reflow. The study conductor edited the wiki in front of the participant so they could notice the changes. This included moving the paragraphs from one point to another. The webpage is then refreshed and the annotations are reloaded, with the ink reflowed (Figure 8 bottom). The participants were requested to rate the system again after seeing this new functionality and asked what they thought of the reflow of the annotation. We also used this data to measure the accuracy of the ink reflow.

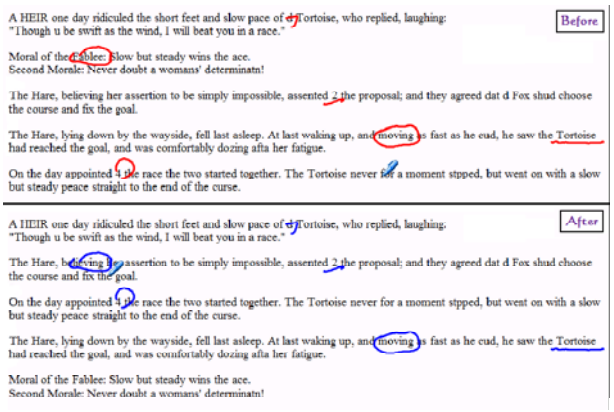

Figure 8: Prototype one user's annotations before and after reflow

The key results showed that the participants found iAnnotate very easy and intuitive to use (4.71 on a 5 point Likert scale). The average accuracy of iAnnotate's reflow functionality is approximately 73%, ranging from 58.33% to 83.33%. This affected the appeal of annotation on the web, which dropped from 4.71 after the first task to 4.42. Some participants mentioned that if they knew how the anchoring algorithm worked, they would be happy to adapt their style of annotating to increase the accuracy of the reflow.

5.2 Prototype Two

The evaluation of the second prototype focused on its new functionality: saving, sending and loading annotations and displaying multiple people's annotations on the one page. The first task was to annotate a web page as an instruction guide to the basics of website layout for an 8 year old (Figure 4). The second was collecting menu choices from friends for a shared meal (Figure 9); we had

pre-prepared four annotations of an online menu site for this.

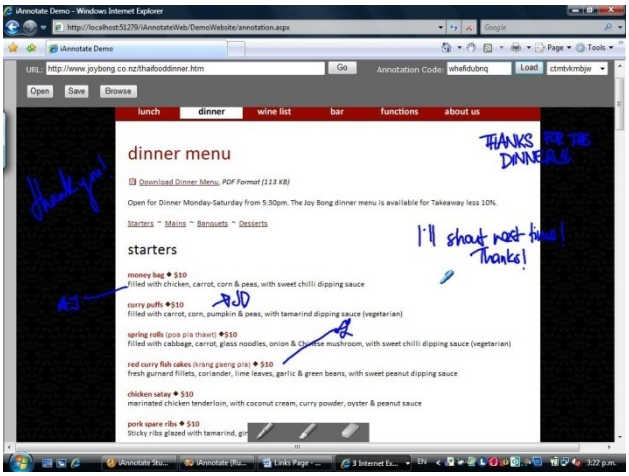

Figure 9: Multiple user's annotations of a web page

The 10 participants were all students (aged 19-26); half were computer science/software engineering majors and the other half were from various other disciplines. The gender split was 50:50 and pen-computing experience varied from a lot (4), some (4), never (2). Each participant was given a short demonstration of the main functionality before they attempted the tasks.

Unlike the first prototype, this prototype exists within the browser pane. The users had to first enter the URL of the web page they wanted to annotate into the text box and press 'load'. Two participants initially put the URL into the browser address bar rather than the iAnnotate textbox and some participants looked for functionality such as loading or saving annotations within the webpage.

The basic functionality of iAnnotate proved easy and, for the most part, intuitive for users (mean 3.6, median 4 on 5 point Likert scale). The save, load and browse buttons were easily located by the majority of users in the study. The participants, even those who had not used a Tablet PC before, all found annotating the webpage to be easy.

The method for saving is likely to cause problems. The user has no choice of what to call the annotation; a random ten character code is assigned when the user saves an annotation. Most of the participants could not recall a code they had saved only minutes before and had to look it up. One participant commented "good but annotation codes were inconvenient". Another suggested thumbnails as an alternative.

A few people interpreted the 'Load' functionality to be equivalent to the 'Open' functionality. 'Load' opens annotations that have been created on the web page currently open and it places them on top of existing annotations. 'Open' opens any annotation created by the user and its associated webpage in a new window. There needs to be a clearer distinction between 'Open' and 'Load'.

We experienced a technical problem during the study with some larger annotations not always loading completely. The program gave no indication of failure. We believe that this was because of time-outs occurring during the round-trip to the server.

A small inconsistency identified was that while a user can input text into the URL box and press Enter to load the corresponding URL, this functionality is not duplicated in the behaviour of the Load tool. When a user inputs an annotation code into the Load box and presses enter, the page is cleared of any previously saved annotations. This may cause loss of data when the page is refreshed which would cause more work for the user and unnecessary frustration.

Most of the usability issues uncovered are not associated with the technical challenges of web annotation and are easy to correct.

The overall impressions of the tool were positive, with most users appreciating the potential usefulness of the product while understanding that this system still needs more development. Feedback included comments such as "fun tool to use" and "good application and has a lot of promise and use".

6 Discussion

In this project our goal was to provide a tool to facilitate digital ink annotation 'anywhere' on 'any' webpage. To this end we have investigated the functionality provided by Microsoft Silverlight as it is a new browser add-in that supports digital inking.

We encountered numerous technical issues, which are not dissimilar to issues previously reported when trying to extend existing proprietary tools (Dietrich, Hosking et al. 2007; Chang, Chen et al. 2008). Nevertheless we had some success with the two prototypes. The first prototype supported inking and basic ink reflow. While the second added saving, loading and display of multiple users' annotations.

The users in both evaluations studies enjoyed the experience and felt that such functionality would be useful. The experience was less compelling with the second prototype as it was a less natural interaction experience with the users having to paste the URL of the page they want to annotate into the annotation site before they could ink, save, send, etc. To partially address this problem we have coded a small JavaScript hyperlink that can reside in the user's favourites that will take the user directly to the annotation server and display the current web page.

While Silverlight offers appropriate base classes for digital ink support, the manner of integration into IE is insufficient for our purposes. The way forward in this respect may be an open source browser so that the base code can be modified as required. It may also be better to use independent inking resources as it is quite probable that proprietary ones have other extensibility issues.

Our goal of being able to annotate any website cannot be fully realized with the current approach. Sites where the content is very dynamic (scrolling adverts for example) are likely to be different on a minute-by-minute basis. Also, there are other web technologies (such as Flash) that can result in multiple pages having the same URL. In this case our recorded URL may not take the user to the appropriate page. To retain information in these cases a copy of the page must be captured – this has obvious consequences for server space etc.

Digital ink reflow remains a difficult problem that has not yet been fully solved. The DOM model of HTML makes this more challenging with these documents. However we expect that as more digital ink annotation research is conducted more intelligent ink reflow approaches will be developed.

7 Conclusions

In this project we have explored general digital ink annotation support in web browsers. Our technology choice was Silverlight and IE. We developed two prototypes, both of which had promising features, but neither of which was a complete solution. To make more progress from a technical perspective requires a change in the browser API or the use of an open source browser. In spite of the technical challenges our usability studies suggest that digital ink annotation of web documents is something that would be useful.

8 Acknowledgements

We thank Microsoft Research Asia for funding this project.

9 References

Bargeron, D. and T. Moscovich (2003). Reflowing digital ink annotations. Chi03, Ft Lauderdale, 385 - 393, ACM.

Bottoni, P., R. Civica, et al. (2004). MADCOW: a multimedia digital annotation system. AVI'04, Gallipoli, Italy, 55-62, ACM.

Brush, A. B., D. Bargeron, et al. (2001). Robust annotation positioning in digital documents. Sigchi'01, Seattle, WA, 285-292, ACM.

Cadiz, J., A. Gupta, et al. (2000). Using Web annotations for asynchronous collaboration around documents. CSCW, Philadelphia, Pennsylvania, 309-318, ACM.

Chang, S. H.-H., X. Chen, et al. (2008). Issues of Extending the User Interface of Integrated Development Environments. Chinz, Wellington, ACM.

Chatti, M. A., T. Sodhi, et al. (2006). u-Annotate: An Application for User-Driven Freeform Digital Ink Annotation of E-Learning Content. ICALT'06, Kerkrade, The Netherlands, 1039-1043, IEEE.

Denoue, L. and L. Vignollet (2001). Annotations in the wild. SAAKM workshop of Semantic Authoring.

Dietrich, J., J. G. Hosking, et al. (2007). A Formal Contract Language for Plugin-based Software Engineering. iceccs, 175-184.

Kahan, J., M.-R. Koivunen, et al. (2002). "Annotea: an open RDF infrastructure for shared web annotations." Computer Networks **39**: 589-608.

Marshall, C. (1997). Annotation: from paper books to the digital library. DL, Philadelphia, 131-140, ACM.

Microsoft Corporation. (2008). "Sliverlight 2.0." Retrieved 20 May 2008, from http://www.microsoft.com/.

Priest, R. and B. Plimmer (2006). RCA: Experiences with an IDE Annotation Tool. CHINZ, Christchurch, 53-61, ACM.

Ramachandran, S. and R. Kashi (2003). An architecture for ink annotations on web documents. 17th International Conference on Document Analysis and Recognition, 256-260, IEEE Computer Society.

Wang, X. and S. Raghupathy (2007). Ink annotations and their anchoring in hetergeneous digital documents. ICDAR, 163-167, IEEE Xplore.

Wilson, M. "Annozilla (Annotea on Mozilla)." Retrieved June 2009, from http://annozilla.mozdev.org/.

Wolfe, J. L. (2000). Effects of annotations on student readers and writers. Digital Libraries, San Antonio, TX, 19-26, ACM.