Nectar: A browser-agnostic contextual web annotation tool

Daniel S. Crosta

December 14, 2005

Abstract

Nectar enables users of the MoinMoin collaborative wiki editing environment to conveniently create and view in-text annotations to wiki documents without disturbing the underlying document. It is designed to facilitate online collaboration for traditional classroom learning by allowing authors to lock documents for editing, but still solicit comments online. Nectar has an obvious and easy to learn yet unobtrusive interface, supports arbitrary threaded conversations started from any annotation, and supports both contextual and page-level annotations.

1 Introduction

For both large lecture-style classes, and smaller discussion-oriented ones, the World Wide Web has proven to be a valuable resource, allowing both teachers and students to extend learning beyond the walls of the classroom. Early, primarily commercial work (eg Blackboard [3]) focused on easing course management for large classes, with tools like gradebooks, online document submission and distribution, and self-grading quizzes. Recently, focus has begun to shift toward supporting collaboration among students in small classes and groups within larger classes on the web. Most educational courseware systems now include at least online discussion forums associated with each class, and some support collaborative editing environments (eg Blackboard's "Team Sites" tool) which allow users to create web documents quickly, without being bothered to learn complex markup langauges.

These tools, however, are not tightly integrated in the online interface, so that discussion boards and collaborative editing tools are often a few clicks away, on a different web page. I believe that for collaborative tools to be widely adopted, they must be available in all parts of the online environment. This allows discussion and collaboration around web documents to take place in context, as discussion of paper documents in class takes place with the documents in front of participants.

One area of active industrial and academic research hopes to bridge the gap between traditional in-class and web learning by enabling students and faculty to create in-text annotations on internet documents. This is usually called web annotation. In particular, Chong and Sakauchi [6] report that "students have the tendency to write everything down during a lecture...for fear of missing something important." As laptop and tablet computers become more widespread, it will be possible for students to have access to prepared lecture notes during the

Proceedings of the Class of 2006 Senior Conference, pages 1–8,

lecture. Web annotation, then, allows students to make their own notes directly "on" the online lecture notes, instead of having to keep personal notes and prepared lecture notes separate, avoiding the "split interface" problem also identified by Chong and Sakauchi.

Web annotation is not a new idea, and several commercial and academic groups have made preliminary stabs at it. Each implementation tackles part of the overall goal, with varying degrees of sucess. This work attempts to draw together insights and advances made in the past eight years, and create an integrated web annotation system for the MoinMoin [12] online collaborative Wiki [10] environment. The Wiki environment, while not specialized for educational use, is a promising platform nonetheless, as its flexibility allows for a variety of uses. I believe that an annotation tool in this environment is more valuable than one specialized for any particular existing courseware system, since it need not be specialized for all the types of documents one might want to annotate.

2 Related Work

As early as 1997, instructors at Freiburg University identified the need to allow students to make annotations to online course material. Bacher et al. [2] describe a case study of moving course materials for an algorithms and data structures class to the world wide web. In it, they identify the need for an analog to marginal notes students would make on in-class handouts. Additionally, they identify the need for "annotations at various levels," where notes may refer to either an entire document, or some paragraph or segment of text from within the document. This notion was later termed "idea-level placement." [5] Unfortunately, they do not describe algorithms nor data structures for dealing with changing underlying document text. In this system, notes are primarily for personal use, though they can be shared on demand.

Cadiz et al. [4] performed a case study of a working online annotation system at a business with over 400 users and over 1,200 documents, in which they discuss the "orphaning" problem of notes losing the context to which they originally refer. To address the problem, they suggest automatic notification to the note's author, who may then choose new context for the note, or determine that it may no longer be relevant to the document.

The SCHOLION [9] system is a distance-learning solution for publishing lecture slides. Students may arbitrarily annotate these slides for personal or shared use. Additionally, students may reply to questions or notes posted by other students by creating links from annotations to a class discussion board area. It is unclear whether the discussion thread contains a reverse link to the original annotation.

Finally, Chong and Sakauchi [6] present the first in-browser web annotation tool geared toward educational needs. Among their design goals, they reiterate the study benefits of annotating course materials. They also wish to support students working in groups either concurrently or asynchronously, both for collaborative projects and group review sessions.

3 Overview and Goals

I believe that a next generation web annotation system ought to support all of the following features, from an end user's perspective:

- *Use context when it makes sense.* Previous work [2, 4, 6, 9] has shown that for many tasks, notes should be created and displayed in context. If a note refers only to a particular sentence, that should be made obvious in both input and display. On the other hand, some annotations may refer to the document as a whole (imagine, for example, concluding thoughts a professor makes on a student's paper). Thus, we have two annotation types: *idea-level* and *page-level* annotations.

- *Annotations* are *conversations.* SCHOLION [9] introduced the notion of linking discussions to annotations. In the Microsoft Office Web Discussions system discussed in [4], this idea is taken further, by putting the conversation in context within the document being discussed. Additionally, the notes should be viewable as conversations in their own right, with *backlinks* to the document which spurred the conversation. A web annotation system should not impose one interface metaphor or the other on users; both should be available.

- *No learning curve.* Using the annotation system should not be an imposition on the user, either by requiring the installation or use of special software (eg a new browser, plugin, or other software), nor with a clumsy or unusual interface. Using standardized web technologies ensures that users will not be confused by the interface, and has the added benefit of supporting nearly all users without any modification to their computer systems. Nectar requires only a CSS- and JavaScript-capable web browser, which is a safe assumption on nearly every computer from which a web annotation system is likely to be used.

These goals, then, inform the design of the Nectar system. The most obvious data structure with which to implement threaded notes is a tree, in which each branch represents a thread of conversation. No restrictions are placed on the branching pattern of the tree. Some notes may comprise only a single node (if there are no replies), while others may spawn deeply threaded discussions, which will take the form of highly branching or very tall trees. Conceptually, the document itself forms the root node, of which each annotation is a child. Replies to annotations form additional leaf nodes, which can become internal nodes when they themselves are replied to.

Idea-level notes store an additional field in the root node: the location in the document that the note references as context. This is used to locate the idea-level notes in the text when displaying the document, and also to gather the context for display alongside the annotation.

4 Implementation Details

The Nectar system is implemented as a set of plugins to the MoinMoin wiki system. The wiki, with its emphasis on shared authorship of web documents, is a solid starting point for developing a collaborative educational courseware system. In addition, MoinMoin handles many of the details of a multi-user online editing environment, such as user authentication and access control. MoinMoin's modular design also allows the wiki formatting markup to be reused within the annotation plugins.

Storage Notes are stored on the web server as XML documents, with one note per file. MoinMoin has a separate directory structure for each page of the wiki, in which Nectar creates a `notes/` subdirectory, so notes are stored, as well as displayed, contextually. This helps offset some of the cost of using the filesystem to store notes, since it is known exactly which set of notes apply to any given document. Each file has a unique filename, generated from the creation date, type, and title of the annotation it contains. The filename type marker further defrays the cost of using the filesystem, as notes need not be read to determine which type of note it contains. The tree structure is a doubly-linked tree, in which each root and internal note contains a list of child note filenames, and each internal and leaf note contains the filename of its parent. This makes it possible to recursively reconstruct the entire thread of conversation from any note, which would be helpful when an internal or leaf note is found as a result of a search.

Storing each note as a file makes the procedure for displaying notes with a document simple (list the `notes/` directory) rather than complex (creating or dealing with a more robust database system). It does not, however, require significant wasted effort to compute note threads, since identifying root nodes requires only listing the directory, rather than reading and parsing the note files themselves. Once the root note is parsed, all the information necessary to create the next level of the tree is present in that file. For this reason, it was only more complex, and not significantly more efficient, to store notes using a heirarchy of folders and files to mirror the tree structure, rather than storing all notes in a single directory.

Storing notes on the filesystem as XML carries with it some overhead in space and time which may be unsuitable for large deployments. In particlar, since each note is stored in its own file, small notes take up more space on disk than they otherwise might, since the minimum actual file size in most filesystems is the block size (often four kilobytes). The average note, on the other hand, even including the verbose XML wrapper, is usually under one kilobyte. In a relational database, the note would only take up as much space as its contents and metadata (creation date, type, context region, etc) require, and the block size isuse is not encountered since all database records are typically stored in a single file. Additionally, searching notes stored in the filesystem would require opening all the files and parsing the XML, which is probably significantly slower than search in a relational database. Implementing acceleration techniques (eg indexing note contents) duplicates much of the effort put into developing database systems. Nectar's storage and interface implementations are kept strictly separate, to facilitate adapting Nectar to use a more robust storage system, should this need arise.

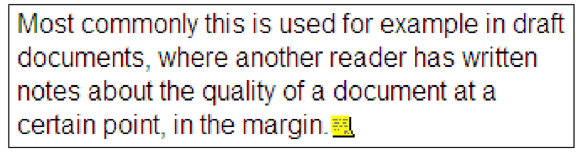

Figure 1: The note icon indicates the presence of an idea-level annotation.

Rendering During page rendering, all the root notes for a page are loaded and parsed. For each idea-level note, an icon is placed in text at the end of the referenced area, to indicate that a note exists. When the user clicks the note icon (Figure 1), JavaScript [7] running in the browser modifies the page's Document Object

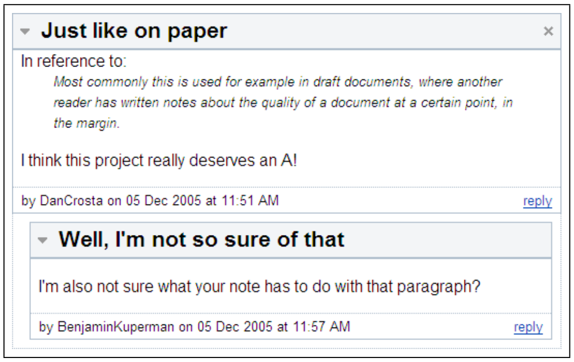

Figure 2: For idea-level annotations, this box displays the threaded conversation floating "above" the document text. The triangle and × icons may be used to collapse or completely hide the notes, respectively. General annotations are displayed similarly below the document text.

Model (DOM) to "pop up" a box in the same page to display the notes (Figure 2). Page-level notes, are displayed at the bottom of the page, and are always present when viewing the page. Both idea- and page-level notes are displayed with a collapsable, threaded view, so that the user may choose which notes and replies to view at any given time.

Creating Notes To create a note, the user has several options. To create new page-level notes, the user fills out a form at the bottom of the page, prompting for a title and comment about the page. Creating an idea-level note is as simple as selecting a portion of the document and hitting the "n" key (for **n**ew or **n**ote, whichever is easier to remember). When a user does so, JavaScript again modifies the DOM to pop up a form similar to that of a page-level annotation, which the user may fill out to create a new idea-level note. The form also displays the text the user selected, for verification (Figure 3). The forms are submitted to the webserver, which updates the note tree data structures, and then redisplays the page to the user.

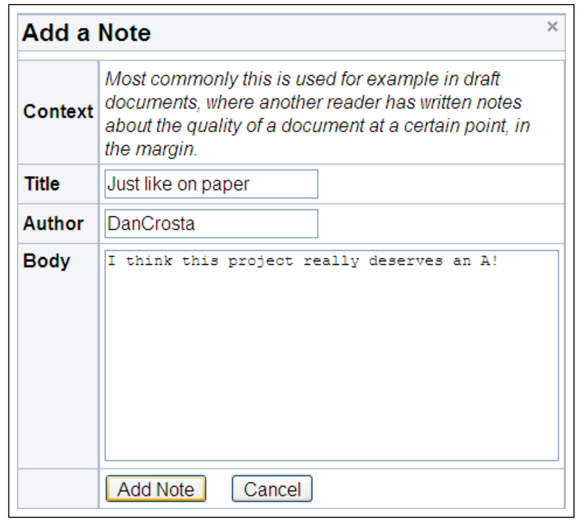

Figure 3: To create new annotations, the user is presented with this prompt for information. The form for creating a new general note is similar, but lacks the "Context" display area.

Detecting the range of text a user selects is accomplished through a bit of JavaScript and CSS [14] trickery. At page render time, the following snippet of HTML code is inserted into the middle of each word in the document:

```
<span class="nw">_nw_N_</span>
```

where N is a running word count in the document. Ordinarily, all the text in the `span`, that is, the `_nw_`N`_` would be displayed, but the CSS stylesheet loaded for the page disables this display. However, since the markers are part of the page text, the JavaScript `getSelection()` method used to find what text the user has selected will contian them. Special care is taken when inserting the invisible markers not to invalidate the XHTML [15] structure of the document, so that it displays properly in all browsers.

A regular expression is used to parse out the first and last markers in the selected text, and these are sent to the server along with the user's form submission. Since the markers are inserted into the middle of each word, the first and last markers are the word indices of the first and last words, respectively, to be more than half selected by the user. A little additional parsing is required to handle single-letter words, but this is straight-forward and not prone to mistakes.

This hidden marker system is not without costs, though. Since the marker itself is significantly longer than the average word length, for long documents (where the length of the document text outweighs the static overhead of wiki headers), the hidden markers may add as much as five to six times more data to transfer. Additionally, the markers are visible in the document source, which may be undesirable to some users. Unfortunately, this is the only system likely to work across the diversity of web browsers currently in use. There are several possible solutions to these problems, discussed in detail in Section 5. To decrease overhead, if a particular user does not have permission to annotate a given document (permissions in MoinMoin documents are set by document editors, in the form of access control lists), none of the hidden markers are sent to the user's computer, nor does the interface for adding annotations (by highlighting) appear.

5 Future Work

The problem of changing underlying documents, identified earlier by [2, 4, 6], remains to be solved. Though Nectar currently does not implement a solution, MoinMoin implements a form of version control similar to that provided by CVS [8]. In particular, changed portions of the document can be identified between any two revisions. With this information, it may be possible to preserve many annotations in changing documents by updating the context markers according to how the document changed. It may also be possible, though more complex and expensive, to use language processing techniques to relocate annotations with a high degree of success based on the contents of the document itself. As a fallback, users should be notified when the document is changed, as suggested in Cadiz et al., so that they may update or invalidate their annotations as appropriate. Currently, Nectar makes the simplifying assumption that documents do not change, even though this is not realistic.

Another major problem, which seems specific to Nectar, is the extra overhead of the hidden markers required for idea-level annotations. Some existing systems [4] allow annotations only at predefined intervals, eg the end of each sentence or paragraph. I believe that word-level granularity or finer is necessary to create a useful analog to pen-and-paper annotation. Another possible solution is to rely on JavaScript to insert the hidden markers in the plain text after it is delivered to the browser. However, this requires a consistent implementation of regular expressions in all browsers, which has not been the case in my experience. Finally, the extra overhead of the hidden markers may become insignificant even in long documents as high bandwidth internet access continues to expand.

Other features that might be added to Nectar are different interaction modes with the annotations. In particular, further integration of the discussion metaphor with the annotations would be ideal. Users should be able to browse annotations either through the document annotation interface Nectar currently implements, or a "forums" view, where annotations are presented as a list of conversations about a wiki page, with appropriate backlinks to the original document. A feature commonly found in web forums is search, which also makes sense for annotations. Both of these features should be straightforward to implement in future versions of Nectar.

6 Post-Mortem Reflections

When I began this project, I wanted to recreate an interface similar to Microsoft Word's [11] "track changes" interface. Unfortunately, the diversity of browsers, all implementing different subsets of different interfaces and technologies, and none of which are geared toward designing interactive user applications, makes this goal nearly unattainable. I ended up doing the lion's share of the work on the server side, for the goal of consistency between browsers and ease of implementation. It is, in many cases, possible to create fully cross-browser JavaScript applications with advanced functionality, but in most cases this requires creating a meta-API where each function your application calls is a wrapper which first detects browser capabilities, and then decides how best to go about the task. Developing such a meta-API requires time I did not have this semester, so at this point, Nectar is only tested and known to work with Mozilla Firefox [13] and Apple's Safari [1]. In future versions, I expect that more of the work can be done on the client side, easing the load on the server and increasing Nectar's scalability.

When comes to the actual code of Nectar itself, I am now beginning to understand the "write it once, throw it away, write it again" mentality in software design. There were some challenges I didn't see ahead of time, in particular, the need for inserting hidden markers in the text. I thought that it was generally possible to determine which DOM nodes were selected by the user, or in which DOM nodes the selection was, which ought to have been enough to figure out selection from the plain document text. This stumbling block alone was enough to delay Nectar by about a week as I struggled first to find out how it was possible ("it *must* be possible, right?"), and then to figure the best way around it. There were a few ways to insert the hidden markers, and it took some time to decide on the middle-of-the-word strategy. This seemed to represent the best mix of minimizing overhead while still allowing flexible and useful annotation.

I also found the MoinMoin API a little messy and undocumented, and in particular often got frustrated at the "TODO: update this comment for 1.3.5" type comments strewn liberally throughout the code. I think free and open source software is a fanstastic idea, and the availability of MoinMoin's source code made this project possible, but I wonder if it is possible in such an organically grown project as MoinMoin to maintain strict standards of coding, testing and documentation.

Finally, I wish that neither of the above had been problems so that I could have had time to tackle what I see as the major advance of Nectar among web annotation systems, the "forums" interface I mentioned in Section 3. It has always been a priority for me, both in this project and in other applications I have made, not to try to impose interaction methods on the user, or at least to avoid doing so as much as possible. In addition to adding interface flexibility, I had hoped that the forums interface would allow users to discover new wiki pages by browsing what was being said about them. It has certainly been my experience in education that the discussion around and about class readings, for instance, is both more in-depth and more comprehensible than the primary document; I suspect this is true for other fields, as well.

References

[1] Apple Computer, Inc. Apple - Mac OS X - Safari RSS. URL `http://www.apple.com/macosx/features/safari/`.

[2] C. Bacher, R. Müller, T. Ottmann, and M. Will. Open hypermedia educational environments: A feasible approach to overcome some difficulties. Technical Report 91, Freiburg University, 9, 1997.

[3] Blackboard, Inc. The Blackboard Academic Suite. URL `http://www.blackboard.com/products/as`.

[4] J. J. Cadiz, Anop Gupta, and Jonathan Grudin. Using web annotations for asynchronous collaboration around documents. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, 2000.

[5] Laurie Causton. Web-based tools for document annotation, 1999. URL `http://www.elpub.org/html/tool_annot.html`. Document no longer available, cited in [6].

[6] Ng S. T. Chong and Masao Sakauchi. Creating and sharing web notes via a standard browser. *ACM SIGCUE Outlook*, 27 (3), September 2001.

[7] ECMA International. Stanrard ECMA-262: ECMAScript Language Reference, December 1999. URL `http://www.ecma-international.org/publications/ECMA\_ST/Ecma-262.pdf`.

[8] Free Software Foundation. CVS - Open Source Version Control. URL `http://www.nongnu.org/cvs/`.

[9] Barbara Froshcauer, Chris Stary, Michael Ellmer, Thomas Pilsl, Wolfgang Ortner, and Alexandra Totter. Scholion: Scaleable technologies for telelearning. In *Proceedings of the 2000 ACM symposium on Applied computing*, volume 1, 2000.

[10] Bo Leuf and Ward Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley Professional, April 2001.

[11] Microsoft Corporation. Microsoft Office Online Home Page. URL `http://office.microsoft.com/en-us/default.aspx`.

[12] MoinMoin. URL `http://moinmoin.wikiwikiweb.de/MoinMoin`.

[13] The Mozilla Corporation. Firefox - Rediscover the Web. URL `http://www.mozilla.com/firefox/`.

[14] World Wide Web Consortium. Cascading Style Sheets, level 2 revision 1 CSS 2.1 Specification, June 2005. URL `http://www.w3.org/TR/CSS21/`.

[15] World Wide Web Consortium. XHTML 2.0, May 2005. URL `http://www.w3.org/TR/2005/WD-xhtml2-20050527/`.