HyberSource: Bridging the Gap Between Source and Code-Related Web Sites

Björn Hartmann
University of California, Berkeley – Computer Science Division
bjorn@eecs.berkeley.edu, mark.dhillon@gmail.com

ABSTRACT
Programmers frequently use the Web while writing code: they search for libraries, code examples, tutorials, documentation, and engage in discussions on Q&A forums. This link between code and visited Web pages largely remains implicit today. Connecting source code and (selective) browsing history can help programmers maintain context, reduce the cost of Web content re-retrieval, and enhance understanding when code is shared. This paper introduces HyperSource, an IDE augmentation that associates browsing histories with source code edits. HyperSource comprises a browser extension that logs visited pages; a novel source document format that maps visited pages to individual characters; and a user interface that enables interaction with these histories.

ACM Classification: D.2.3 [Software Engineering]: Coding Tools and Techniques – Program Editors.

General terms: Documentation, Human Factors

Keywords: augmented source code, browsing history

INTRODUCTION
When programming, developers spend a significant amount of time outside their Integrated Development Environment (IDE) and inside their Web browser. In a lab study by Brandt, programmers spent 19% of their time on the Web [2]; in Goldman’s field study, 23% of Web pages visited in temporal proximity to code edits were development-related [4]. Work-related uses of the Web include finding libraries and source code examples, consulting online documentation, forums, and question-and-answer sites, and managing project-related communication in software forges (e.g., Google Code).

Despite this strong connection between Web browsing activity and code production, today’s IDEs and browsers remain largely unaware of each other. We hypothesize that making the connection between source code and visited pages explicit can benefit programmers in multiple ways. First, for maintenance of one’s own code, access to previously consulted resources can help re-establish context and shorten the time spent re-retrieving previously used resources (a common pattern in Web browsing [8]). Second, for code shared between developers, relevant Web resources can help project newcomers understand design decisions and become productive faster [3].

To investigate these potential benefits, we are developing HyperSource, an IDE augmentation that associates browsing histories with source code edits (Figure 1). Conceptually, HyperSource builds upon read and edit wear [5] by collecting and showing sets of Web pages that were read while code was edited. HyperSource enables this association via a novel source document format that attaches metadata to text at the character level. Annotations are maintained through subsequent document edits (insertions, deletions, copy, cut, paste, undo and redo). As programmers navigate their source, a panel inside the IDE displays associated Web histories and enables interaction with them (Figure 2).

SHARING ANNOTATED CODE WITH HYPERSOURCE
Joe, a Computer Science student, is writing a visualization of Web server logs for his student organization. As he writes his code, he discovers a bug in a drawing library. An online forum post suggests a counterintuitive workaround. As he consults these Web pages, their URLs are tracked inside his IDE and attached to subsequent code edits.

The following semester, Jan, another member of the club, has been tasked with changing Joe’s code to add richer visualizations. She is initially confused by the drawing code since it does not match the library’s official documentation. Because Joe wrote the code with HyperSource, Jan can select the confusing statement and see links to the original forum posts Joe consulted. She thus uncovers the rationale for Joe’s code and can complete her task quickly.

HYPERSOURCE ARCHITECTURE
Tracking and visualizing visited pages requires three components: a browser extension that captures visited Web
pages; an augmented source document data structure that maintains associations between code and pages; and a user interface that shows the collected history for a code selection and enables interaction with this history.

Whenever a new page is loaded in the browser, the HyperSource browser extension sends URL, title, and a thumbnail of the page to the code editor. Our prototype extends Google Chrome and uses XML-HTTP requests.

Inside the IDE, the source document maintains a list of visited pages for every character. HyperSource models user behavior as a cycle of alternating edit and browse phases: code is annotated with the set of pages viewed just prior to the current set of edits. During browsing, visited pages are added to a queue; when the user edits code, all pages in the queue are assigned to all new characters. When the user switches back to browsing, the queue is cleared (Figure 3). We are currently extending this algorithm to consider edit locality, as not all edits are related to browsed pages.

After Web pages are assigned to code, subsequent code edits may insert or remove text, duplicate text through copy and paste, and reverse previous operations through undo and redo. HyperSource maintains associations between characters and pages with position tracking: each character is identified by a position that describes its location in the document throughout edits. While the absolute location of a character changes, it remains identified by the same position object. Position objects are updated after every edit.

The history view (Figure 2) shows two types of information: the queue of pages visited since the last document edit (so users can see what will be saved during the next edit) and a log of previously visited pages for the currently active line or selection. The view updates automatically as the document cursor moves through the source document.

Logged items have varying levels of relevance. HyperSource renders recorded pages with different levels of detail, based on the following metrics: more detail is shown if text was copied from the page into the source document; the developer marked the page in the browser extension; the page was the final page visited before editing.

Not all visited Web pages should be logged. To prune the set of logged pages, HyperSource offers three filtering mechanisms: a blacklist of domain names; a “clear buffer” action that empties the internal queue; and widgets for individual log item removal.

RELATED WORK

CodeTrail [4] most closely matches our motivation of connecting browser and IDE. It automatically selects and attaches documentation to projects using known documentation sites; opens attached documentation in a browser when a source file is opened; and enables users to manually add URL bookmarks to files. HyperSource differs from CodeTrail in focus: it keeps track of visited Web pages without requiring explicit bookmarking and attaches these Web histories to specific sections of code.

Better code search engines, e.g., Assiemi [6], integrate search for documentation and example source. Such tools are not aware of the developer’s IDE. Blueprint [1] incorporates interactions for finding and integrating example code directly into the IDE. In contrast, with HyperSource developers continue to use their Web browser, but gain history capture. HyperSource is also related to research that records Web browsing activity, e.g., ActionShot [7]. We expect to leverage insights on session summarization from this research. Finally, HyperSource is related to IDE enhancements that attach metadata to source code. Hipikat [3] is an early example of extracting data from revision histories. It recommends documentation and issue reports based on source code selections. HyperSource focuses on a different set of metadata (Web pages) and a finer level of granularity (individual editing operations within the IDE).

FUTURE DIRECTIONS: EVALUATION, BEYOND CODE

Our future evaluation seeks to quantify the productivity benefits of using HyperSource and determine the effectiveness of automatic URL collection. In ongoing work we also examine how to apply the described approach to other document authoring tasks beyond programming.

REFERENCES


Figure 3: State machine for tracking browsing & editing.