cTed: Advancing Selection Mechanisms in Web Browsers

Philipp Eichmann
Hyun Chang Song
Emanuel Zgraggen
Brown University
Providence, RI 02912, USA
peichmann@cs.brown.edu
hyun_chang_song@cs.brown.edu
ez@cs.brown.edu

Abstract
Selecting fragments of content on websites, such as text, lists, tables and images, and copying them to note-taking applications or word processors is a common task for information workers. However, web browsers only offer crude support for selections due to the restrictive underlying HTML/CSS model, making it difficult and sometimes even impossible for users to select and copy content. In this paper, we present cTed, a web browser plugin that allows for selection gestures to be drawn directly onto websites. Our plugin intelligently maps selection gestures to underlying HTML content and thereby enables users to more intuitively mark and extract regions of websites while preserving textual and semantic information.

Author Keywords
web browser; selection; gesture; HTML; CSS; interaction design; user interfaces

ACM Classification Keywords
H.5.2 [Information interfaces and presentation]: Miscellaneous

Introduction
Information workers who research topics on the Internet typically visit various websites, bookmarking relevant sources and copying pertinent material to applications such as Mi-
OneNote or Word. To bring fragments from a website to another application, a user must first make a selection. However, selection mechanisms offered by modern web browsers are limited to defining a start and end point, which are then mapped to a sequence of consecutive HTML elements. Since the appearance of a rendered website does not necessarily reflect the underlying HTML/CSS model, the resulting selection determined by the web browser can diverge significantly from a user’s intention. Consider Figure 1, for instance, where a user tried to select a block of cells within a table (columns “1930” to “1960” in rows “California” and “Colorado”) by dragging the mouse from the start to the end cell. The web browser’s restrictive HTML/CSS based selection model does not allow for such selections and simply highlights all cells between the start and end cell defined in the underlying HTML structure.

Related Work
The importance of annotations for true semantic webs [8] has been discussed extensively [14, 10]. Similarly, there is an abundance of literature pointing out the benefits of annotations for reading comprehension [18] and for refining information [16]. The research community has built and evaluated several web annotation systems [9, 15, 12] based on the premises of these findings. These systems offer fine-grained annotations, different annotation types as well as storing, sharing and collaboration features. However, all of them rely on standard browser selection techniques in order to extract HTML/CSS content from web pages.

In the past, many mashup tools (e.g., [11, 17, 13]) and browser extensions [3, 1, 5, 2, 6, 7] have emerged to address the shortcomings of web browsers’ content extraction capabilities. However, like the web annotation systems above, all of these tools rely on standard browser selection techniques when selecting fragments of a website. Users often resort to making screen shots using an operating system feature, a third-party application or an integrated technique offered by some of these extensions. It allows them to define a rectangular region, whose underlying area of
A website is projected onto a flat bitmap image. Although this method gives users more freedom in defining a region they intend to select, much of the textual and semantic information provided by the underlying HTML/CSS content is lost. While some of the content can be recovered (e.g., using optical character recognition for text), information hidden in HTML/CSS such as a link, emphasize or list/table semantics is irreversibly removed.

Native browser support for annotations on websites beyond simple bookmarking has only recently begun to emerge. Microsoft for example, incorporated some of this functionality directly into its Edge browser, which has been released with Windows 10. The Edge browser introduces a feature called “write on the web” [4] that allows users to capture, draw, highlight, and make notes directly through the browser. However, the only way HTML content can be selected and extracted remains unchanged and still requires a start and end point, which are then mapped to a range of consecutive HTML elements.

**cTed**

In the following we describe how gestural selections can be implemented using just JavaScript and the DOM API, making our technique portable to most web browser plugin architectures. We present three different selection gestures (Figures 2, 4, 5) and compare our results to the native selection mechanisms of Google Chrome in Figure 3.

**cTed**, our plugin for Google Chrome extends the web browser’s functionality by injecting custom JavaScript into a website. In order to allow for gestural selections to be drawn on to an HTML website without interfering with the browser’s own functionality, it first appends a fixed-positioned, transparent canvas element to the DOM and makes sure it appears as top most element in the visual.

**Line Selection**

A line selection (Figure 2) is the simplest of our three supported selection gestures and mimics the native selection mechanism, where a start and end point is defined using the cursor. Finding the HTML elements along a drawn line is achieved by first converting the left-most point (start) and the right-most point (end) of a line to a range object using `caretRangeFromPoint`. This gives us access to the start and end text element in HTML as well as the start and end character offsets that allow us to retrieve their inner content. To find all elements that are located between the start and end node, we simply iterate over all siblings of the start node until we reach the end node.

**Bracket Selection**

A bracket selection (Figure 4) is initiated by drawing a vertical line along the left side of content elements, such as paragraphs, images or tables. To find the desired element in the DOM, we sample the rectangular area that spans from the bracket to the right of the browser’s viewport with a step size of 10 pixels in vertical and horizontal direction. For each sample point \((x, y)\) we use JavaScript’s `elementFromPoint` function in order to find the topmost HTML element under the given point.

Since the ink canvas is the topmost element for all points as long as it is shown, we simply remove it from the DOM while sampling. Then, for each sampled point, we traverse the DOM up until we reach an ancestor element that is not fully encompassed by the bracket. All of those elements are assigned a weighted score based on their horizontal location relative to the bracket. This ensures that elements that appear to the right of a bracketed element are given less weight and thus will not be included in the selection. After computing the total score for all elements in the sampling area, we ensure that when multiple consecutive elements are bracketed, all of them will be added to the selection. We do this by com-
**Figure 3:** This figure shows how our bracket and marquee selection algorithms outperform the selection made in Google Chrome (depicted in the native column - the brackets mark the start and end point of the selection with a mouse). Even though the start and end points are set correctly, Chrome spuriously also selects the image and its caption on the right, whereas our simple bracket and marquee gestures are correctly mapped to desired title and paragraph.

Putting the standard deviation of all scores and select only elements that lie within the range of maximum score and the maximum score minus twice the standard deviation.

**Marquee Selection**
A marquee selection (Figure 5) supports content extraction of an arbitrary rectangular area and is triggered when the cursor is moved diagonally across the canvas, marking the top left and bottom right corner of a rectangle. We first find the element that lies underneath the top left corner using `elementFromPoint` and store it in an array of candidate elements. We then find the dimension of that element and search for the next element along the horizontal axis, followed by the vertical axis. All elements are added to the candidates array. We then find the common ancestor of all elements, meaning the smallest element that encompasses all elements within the selection boundaries. Once found, we recursively traverse all of its children to determine their relationship to the selected area. There are three possible cases: (1) an element is fully contained, (2) partially contained, or (3) not contained by the selected area. In the first case, we preserve the element and all of its children. In the second case, we preserve it and recursively visit its children to test for an overlap with the selected area. In the third case, we remove it from the candidates. After all elements have been processed, all remaining candidates are mapped back to their original location in the DOM in order to highlight the nodes’ content.

**User Interface**
cTed can be launched in any tab by pressing the icon in the top right corner of Chrome. Our system automatically...
classifies the user drawn gesture as either line, bracket or marquee. cTed does not require any remote services and runs completely locally. After a selection is made on a website, cTed stores the extracted HTML content in Chrome’s local storage along with additional meta-data such as the current website’s URL, user defined tags entered through the menu, a timestamp etc. A list of all selections made on the currently active website can be viewed by hitting the expand button in the menu (see Figure 6).

A searchable list of all selections independent of their originating website is displayed in a new tab when clicking the “View All” button (7). For instance, an information worker who gathers information on Native Americans could quickly find all relevant selections she has made across multiple websites using search and filtering. More notably, cTed’s ability to restore previously made selections when re-visiting a website essentially surpasses the web browser’s bookmarking functionality, where only links to entire pages are stored and therefore makes it easy to re-visit only the relevant parts of a website.

Discussion and Future Work
We showed why contemporary browser selection mechanisms are insufficient for effective content selection and extraction, and proposed novel gesture-based selection mechanisms. Mapping selection gestures to HTML elements is inherently difficult because the structure of an HTML document can differ severely from its rendered, visual representation. We intend to expand on our current selection gestures and further improve the accuracy and performance of our algorithms. To improve our method we aim to test our mapping algorithms on a large number of websites as well as to incorporate computer vision algorithms into our selection-to-content mapping procedures.
Our plugin allows users to manage, search and filter their selections. Unlike Pinterest, cTed is not just limited to image clippings and could therefore be used to allow users to share any kind of websites fragments among a community of users. A small set of initial pilot studies with students showed that copy-pasting with web browsers is an ubiquitous problem and all participants indicated large interest in using cTed in day-to-day life. We are therefore planning to conduct a rigorous user study and release cTed as an extension in the Chrome Web Store.

Acknowledgements
All web browser and content screenshots were taken on http://www.wikipedia.org using Google Chrome. We would like to thank Bob Zeleznik, Andries van Dam and all reviewers for their valuable feedback and input.

References
