# Zoetrope: Interacting with the Ephemeral Web

*Eytan Adar*[†]*, Mira Dontcheva*[‡]*, James Fogarty*[†]*, and Daniel S. Weld*[†]

[†]Computer Science & Engineering
DUB Group, University of Washington
Seattle, WA 98195
{eadar, jfogarty, weld}@cs.washington.edu

[‡]Advanced Technology Labs
Adobe Systems
San Francisco, CA 94103
mirad@adobe.com

**ABSTRACT**

The Web is ephemeral. Pages change frequently, and it is nearly impossible to find data or follow a link after the underlying page evolves. We present Zoetrope, a system that enables interaction with the historical Web (pages, links, and embedded data) that would otherwise be lost to time. Using a number of novel interactions, the temporal Web can be manipulated, queried, and analyzed from the context of familar pages. Zoetrope is based on a set of operators for manipulating *content streams*. We describe these primitives and the associated indexing strategies for handling temporal Web data. They form the basis of Zoetrope and enable our construction of new temporal interactions and visualizations.

**ACM Classification:** H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

**Keywords:** Web, temporal informatics, Web extraction

**INTRODUCTION**

The World-Wide Web is highly dynamic, evolving with surprising speed over time. News stories, Wikipedia articles, social network profiles, and corporate pages are continually updated. Automatically generated pages, whether driven by back-end inventory-control systems (e.g., travel and e-commerce sites) or real-time sensor streams (e.g., traffic, weather, and webcams) may change even more quickly.

In contrast, most Web *usage* is restricted to the Web's most recent state. People access the Web through browsers that, with the exception of simple caching, provide access only to the Web's present state. Although search engines provide quick access to a vast corpus, retrieval is again limited to the most recent snapshot recorded by the underlying crawler. The headlines on today's CNN homepage will be gone tomorrow, and yesterday's price for a book on Amazon is likely irretrievable today. Even if historical data still exists, it may be aggregated into averages or split among many webpages. For a person who does not have access to historical data, or would find it difficult or impossible to recreate, the Web is primarily one-dimensional, containing only one version of any page. The current lack of support for temporal access to the Web makes it difficult or impossible to find, analyze, extract, and compare historical information.


*UIST'08,* October 19–22, 2008, Monterey, CA, USA..


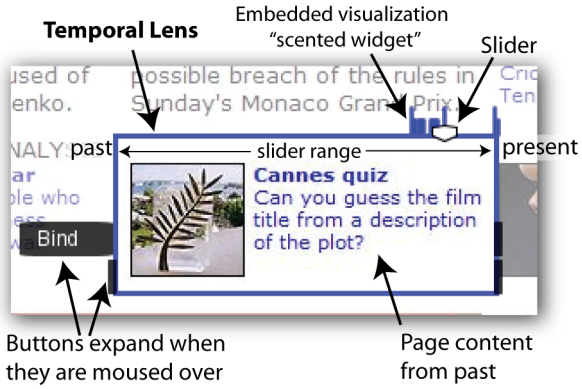


Figure 1: Temporal lenses include a slider for accessing previous versions, a scented widget that shows the location of available content, and buttons for creating visualizations and binding lenses together.

This paper presents Zoetrope, a tool for interacting with the extended history of the World Wide Web. With Zoetrope, one can explore past versions of entire pages or place *lenses* (Figure 1) on any part of a webpage to scan backwards through time (e.g., to see all previous top headlines on CNN's page). Lenses can be *filtered* with keyword and other queries or *bound* together to explore correlated historical data. To support analysis of this data, Zoetrope provides a number of *visualizations* for the extracted content. Because temporal Web data is not currently available at fine granularity, Zoetrope also includes a set of crawling and indexing technologies for obtaining high-resolution temporal data. We use an experimental dataset of 250 webpages from a range of domains and websites (news, sports, weather, entertainment, etc.) crawled at one-hour intervals for more than a month.

By providing a variety of lenses, filters, and visualizations that can be combined and composed in powerful ways, Zoetrope presents a new way to think about historical Web data. As a tool, Zoetrope allows anyone to explore the Web over time. Furthermore, Zoetrope provides developers a mechanism for easily exploring new applications and interactions. The contributions of our work include:

- a novel visual programming toolkit and a set of interactions for rapidly building and testing temporal Web queries,
- a semantics for temporal data streams,
- a set of recomposable operators to manipulate temporal data streams,
- indexing structures for fast processing and interaction with Web content over time,
- and a unique dataset collected by a new crawler design.

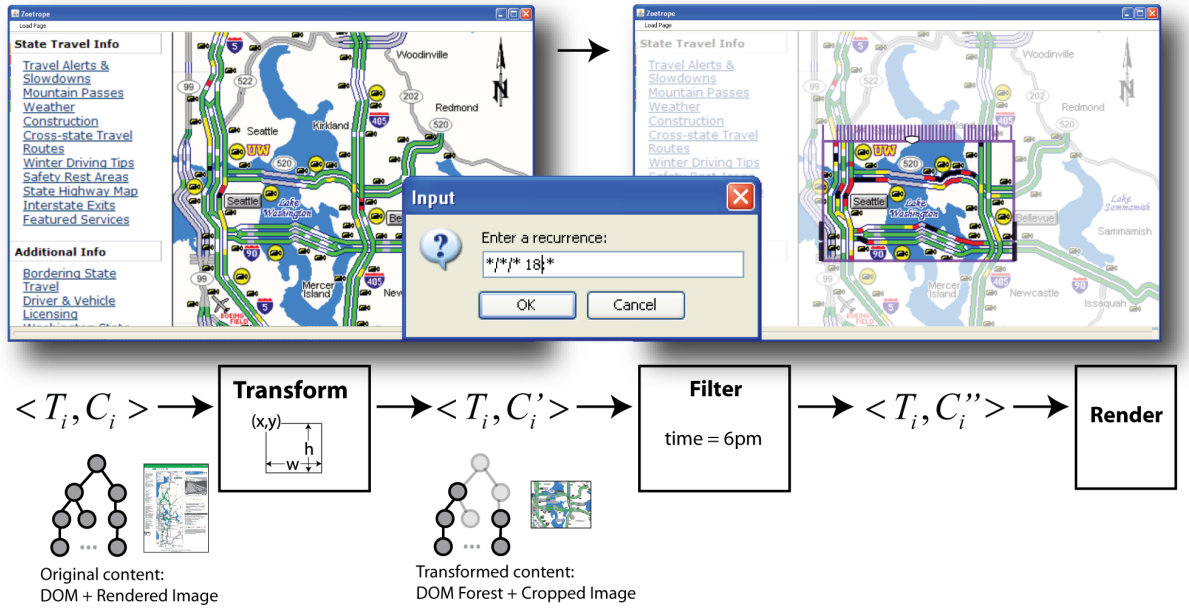


Figure 2: The left browser window shows a webpage for Seattle traffic conditions. A person draws a lens on the region of interest (the bridges crossing Lake Washington) and specifies a temporal filter, selecting only content fetched between 6-7pm (i.e., 18-19 in 24 hour format). The browser window on the right shows the resulting lens, in purple, with tick marks on top indicating each instance of 6pm (the page is faded in the figure to emphasize the lens). Below the browser windows we show the Zoetrope transform and filter pipeline that enable this interaction.

## ZOETROPE INTERFACE

Zoetrope presents webpages in a zoomable canvas, within which a person can simultaneously explore past versions of any number of webpages. To browse previous versions, one can move the slider at the bottom of the display or, alternatively, place one or more lenses on different parts of a webpage. For example, consider Ed, who at lunchtime becomes anxious about his evening commute. Although he knows the traffic page, the page only displays the current traffic conditions. Without Zoetrope, Ed might be able to find older traffic flow maps, but this search would likely require considerable effort. Instead, Ed can adjust the Zoetrope slider and instantly view previous versions of any page. To focus on a specific part of a webpage, Ed selects a *visual* lens from the lens menu and draws directly on the webpage to specify which part of the traffic map is of interest (see Figure 2). He also adds a *temporal filter* to the lens, because he is only interested in traffic information at 6pm. Zoetrope creates a visual lens for Ed's selection that includes a slider, a scented widget [28], and several buttons for further refinement. By moving the slider, Ed is able to shift backwards and forwards day-by-day to review past congestion. Buttons to the left and right of the lens allow Ed to synchronize the entire webpage to a selected time, to close the lens, to bind multiple lenses together, or to generate a visualization based on the selected content. Throughout this task, Zoetrope allows Ed to use a webpage he is familiar with, in its present version, to contextualize his query into the past. Although Zoetrope displays the *rendered* webpage as an image, the system maintains the interactivity of the *live* webpage. Clicking on a hyperlink opens a browser for the present version of the hyperlink target (or the historical version corresponding to the slider selection, if it exists within Zoetrope).

## SYSTEM ARCHITECTURE

The Zoetrope architecture, summarized in Figure 3, includes a *Web crawler* that collects data, a set of *databases* to store collected content, and an *interface* that provides access to the stored data. The crawler collects data at regular intervals and associates each crawled page with a time. Each page is stored in two databases: (1) as XML providing Zoetrope access to the structure and text for indexing and extraction, and (2) as an image providing Zoetrope nearly instant access to the page's visual appearance. When a person views a page in Zoetrope, each instance of the crawled page and associated time are loaded as a *content stream*. Conceptually, a content stream is a sequence of tuples (i.e. pairs), $\langle T_i, C_i \rangle$, where $C_i$ is a *content item*, such as a webpage or some piece of a page, and $T_i$ is the time when that content was sampled from the Web. When a person creates a lens or some other visualization, Zoetrope generates a sequence of *operators*, which process data with transforms and filters, and routes the content stream through them. Finally the stream is displayed using *renderers*. Our current implementation supports *lenses* and *visualizations*, but we expect that there may be other interesting types of renderers. Additionally, Zoetrope can

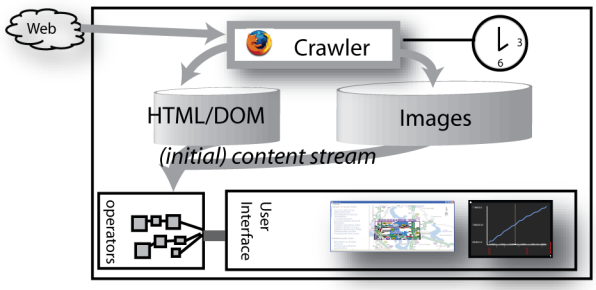


Figure 3: An overview of Zoetrope's architecture.

export content streams to external systems, such as Google Spreadsheets (*docs.google.com*).

In Figure 2, for example, we see a graphical representation of the operators corresponding to Ed's visual lens: a transform consumes the content stream, it extracts a small cropped image and a small part of the document text, a filter decides which instances occur between 6 and 7pm, and the renderer uses the slider location to display a specific instance from this stream.

## TEMPORAL LENSES

A lens allows a person to select some content and track it over time. Although there are various flavors of lenses, their creation and use is nearly identical. A person creates a lens simply by drawing a rectangular area on the webpage surface. In the underlying semantics of the Zoetrope system, the creation of a lens produces a parametrized transform operator that acts on the original page content stream, an optional filter (or set of filters) that processes the transformed stream, and a renderer that displays the historical data in the context of the original page (see Figure 2). The specific selections of transforms and filters depends on the lens type. We currently implement three types of lenses for tracking different types of information: visual, structural, and textual.

### Visual Lenses

The simplest Zoetrope lens is the visual lens. To create this type of lens, a person specifies a region on the original page (e.g., a portion of the traffic flow map in Figure 2). The specification produces a lens with a slider. The slider is parametrized on the width of the lens and the range of data being displayed. As the slider moves, the lens renders the corresponding data.

More formally, the lens specifies a rectangle $R$ (coordinate of origin, width, and height), which is used to parametrize a cropping transform. The cropping transform consumes a stream of Document Object Model (DOM) trees and rendered page images and outputs a stream of DOM forests (the subset of the original DOM that is rendered within the rectangle) and cropped images. Recall that a content stream is represented as a set of tuples, $\langle T_i, C_i \rangle$, where $C_i$ is the content and $T_i$ the time. In this case, $C_i$ contains the document (as a DOM tree), the CSS templates, images, and flash files (call these $C_i.\text{DOM}$, $C_i.\text{CSS}$, etc.) As an optimization, the tuple content also contains the rendered representation of the page, $C_i.\textit{img}$. Thus, a visual lens converts each tuple of the form $\langle T_i, C_i.\text{DOM}, C_i.\textit{img} \rangle \rightarrow \langle T_i, \text{SELECT}(C_i.\text{DOM}, R), \text{CROP}(C_i.\textit{img}, R) \rangle$. The generated stream is passed to the lens renderer, which displays the cropped image corresponding to the selected time.

### Structural Lenses

Not all webpages possess sufficient stability for a visual lens. Slight shifts in rendering or more significant movement of elements can cause distracting jumps when a person moves the lens slider. To counter this effect and to allow for more precise selections, Zoetrope provides structural lenses. Structural lenses are created in the same way as visual lenses, by drawing a rectangle around an area of interest, but they track selected HTML content independent of visual position.

Formally, $R$ denotes the rectangle that identifies the DOM elements of interest at lens creation time, $s$. The lens creation process finds the XPath, $P$, that describes the upper-leftmost element rendered within $R$. When the path $P$ is applied to the DOM at time $s$, $\text{XPATH}(P, C_s.\text{DOM})$, the result is some DOM element, $E_s$. We use $E_s.x$ and $E_s.y$ as the origin coordinate for the crop *in all other versions* and are able to track selections more accurately. A precise specification of $R$ is not necessary. Zoetrope finds all DOM elements contained in the selection and resizes the rectangle (now $R'$), visually snapping to the selected elements.

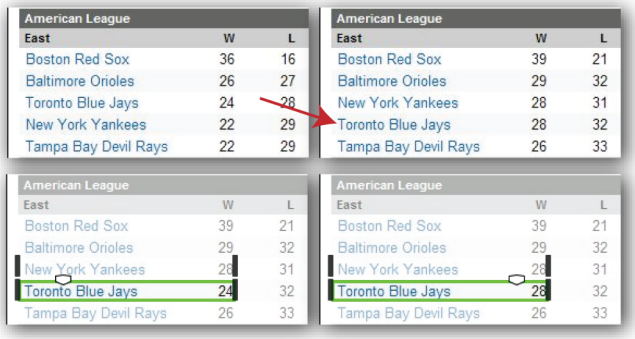


Figure 4: A textual lens can track content regardless of where it appears on the page, such as the Toronto Blue Jays, which over time shift in the ordered list above.

The transform works on tuples in the following way:

$$\langle T_i, C_i.\text{DOM}, C_i.\textit{img} \rangle \rightarrow \langle T_i, \text{SELECT}(C_i.\text{DOM}, R'), \text{CROP}(C_i.\textit{img}, R'' = (E_s.x, E_s.y, R'.\textit{width}, R'.\textit{height})) \rangle$$

As before, this output stream of tuples is sent to a lens renderer which displays the image for the selected time.

### Textual Lenses

Visual and structural lenses are dependent on certain types of webpage stability. A visual lens relies on stability of the rendering, whereas a structural lens takes advantage of structural stability. Both are reasonable in many scenarios, but it is also worth considering selections based on unstable or semi-stable content. For example, consider tracking a specific team in a list of sports teams that is ordered by some changing value, such as rank (see Figure 4). As teams win and lose, the team of interest will move up and down the list. Specifying a rectangle at $(100, 400)$, or the fourth row in the list, will not work when the team moves from this position. To address this type of selection, we introduce the notion of a textual lens, which tracks a textual selection regardless of where the text is located on the page. A textual lens can track exactly the same string (e.g., a blog story) or approximately the same string (e.g., a sports team name with a score, where the score changes from time to time).

In its most general form, a textual lens tracks arbitrary text regardless of where it appears on the page, which DOM elements contain the text, and the size of those elements. This generalization is unfortunately too computationally intensive, even in scenarios where the text is unchanging, and the problem becomes intractable for an interactive system. To make our textual lenses interactive, we restrict the search space by making use of the document structure. Textual lenses often track items that appear in tables,

lists, or structurally similar sub-trees (e.g., posts in a blog all have similar forms). We take advantage of this structural similarity by generalizing the paths of selected DOM elements. For example, a path of the form `/html[1]/body[1]/table[4]/tr[5]/td[2]` can be generalized to `/html/body/table/tr/td`. Applying this more general XPath to the DOM returns all elements that match this pattern, instead of just the specific item that was selected. In practice, we find that this transformation overgeneralizes the selection and that CSS class attributes are useful for constraining the patterns to eliminate unlikely DOM elements. For example, the headline and body of an article may both match the pattern above (they are both in a table), but the headline will have "headline" as its class attribute while the article will have "article" as its class attribute. We therefore add class attributes to the generalized path (e.g., `/html/body/table/tr/td [@class='headline']`) to constrain the resulting set of similar items. Given this generalized path, $P'$, the transform produced by the lens identifies all matches at a particular time by applying the path to the DOM (i.e., $\text{XPATH}(P', C_i.\text{DOM}) = \{D_{i,1}, \ldots, D_{i,n}\}$).

Next, Zoetrope finds the closest matching element for each time using a similarity metric, SIM, which calculates the textual similarity of the originally selected content, $D_s$, to each of the extracted elements. For each time, $i$, Zoetrope finds the $D_{i,best}$ that is most similar to $D_s$. We currently use the Dice coefficient to compute similarity, which calculates the overlap in text tokens between two pieces of text (i.e., $\text{SIM}(A, B) = 2 * |A \cap B|/(|A| + |B|)$, where A and B are sets of words). When the initial selection is a forest rather than a tree of DOM elements, Zoetrope generates the cross product of each generalized path. In other words, given two generalized paths, $P^1$ and $P^2$, we calculate

$$\begin{aligned}&\text{XPATH}(P^1, C_i.\text{DOM}) \times \text{XPATH}(P^2, C_i.\text{DOM}) = \\ &\quad \{D^1_{i,1}, \ldots, D^1_{i,n}\} \times \{D^2_{i,1}, \ldots, D^2_{i,m}\} = \\ &\quad \{\{D^1_{i,1}, D^2_{i,1}\}, \{D^1_{i,1}, D^2_{i,2}\}, \ldots, \{D^1_{i,n}, D^2_{i,m}\}\}\end{aligned}$$

These groups are filtered to find those that satisfy the initial hierarchical relationship of the selection. For example, if two sub-trees in the original selection have a hierarchical distance of 5, the pairs tested in the generalized path should have the same distance. An alternative to hierarchical distance is visual distance (i.e., both elements should fit inside a rectangle of the size originally specified by the lens). Similarity in such cases could be determined as a combination of similarity scores. Our initial implementation simply merges all the text from a group into one text block before calculating the similarity. A threshold can be set as input to the transform, restricting returned elements to those sufficiently similar to the input. This threshold can be varied depending on how much the content is likely to change from the original selected version.

To summarize, a textual lens is created by the selection of a region, the automatic conversion of the selection to a generalized path, and the use of this path as a parameter to the transform. The transform works on the DOM and images for each time, generating an image cropped to the upper-left coordinate of the best matching element set ($D_{i,best}$) and sized to the width and height of the original selection, $R$.

$$\begin{aligned}\langle T_i, C_i.\text{DOM}, C_i.\textit{img}\rangle \rightarrow \langle T_i, D_{i,best}, \text{CROP}(C_i.\textit{img}, R'' = \\ (D_{i,best}.x, D_{i,best}.y, R'.width, R'.height))\rangle\end{aligned}$$

As before, the output is passed to a lens renderer that displays the appropriate cropped image according to the time selected with the slider.

Taken together, the visual, structural, and textual lenses form the basis of Zoetrope's interactions. They are sufficient for tracking and extracting a wide range of data types within a page, thus enabling Zoetrope's filters and visualizations.

### Applying Filters to Lenses

Given the large volume of data encoded in a content stream, it is natural to want to focus on specific information of interest. Zoetrope uses *filters* to provide this capability. If we consider the evolving state of content as a database relation, then a filter is simply a select operation in relational algebra (e.g., *select* tuples *where* some condition). There are a number of useful conditions for selection, including:

- **Filtering on Time:** As in our original example, one may wish to see the state of one or more streams at a specific time or frequency (e.g., 6pm each day).
- **Filtering on a Keyword:** The selection condition may also refer to $C_i$, the content half of the tuple $\langle T_i, C_i\rangle$. If $C_i$ contains text, then keyword queries may apply. For example, one might only be interested in headlines that contain the word "Ukraine" (Figure 5).
- **Filtering on Amounts:** One may also select content using an inequality and threshold (e.g., $> k$). If the content is numeric and the inequality is satisfied, then the tuple is kept; otherwise it is filtered. Similarly, one can select the maximum or minimum tuple in a numeric stream.
- **Duplicate Elimination:** It may also be useful to select only those tuples whose content is distinct from content seen earlier in the stream.
- **Compound Filters:** Logical operations (conjunction, disjunction, negation, etc.) may be used to compose more complex selection criteria.
- **Trigger Filters:** An especially powerful filter results when one stream is filtered according to the results of another stream's filter. For example, Ed can filter the traffic page using a conjunction of the 6pm time constraint and a trigger on the ESPN page for the keyword "home game." We will return to this when considering lens binding.

Because filtering is useful for many tasks, it is provided as an option whenever a visual, structural, or textual lens is applied. When selecting a region with filtering enabled, a lens is created based on the underlying selection and a popup window asks for a constraint to use in the filter, such as a word or phrase. Other appropriate constraints include maximum, minimum, and comparison operators.

Time-based filters can include a time range (e.g., the weather between June and March) or a sampling rate (e.g., the ending stock price every day). To use a time range, a person must specify a start time and end time. Sampling rates are specified with recurrence patterns of the form: `Month/Day/Year Hour:Minute`. For example, a lens with time filter `*/1/* 18:*` displays content from the first day of

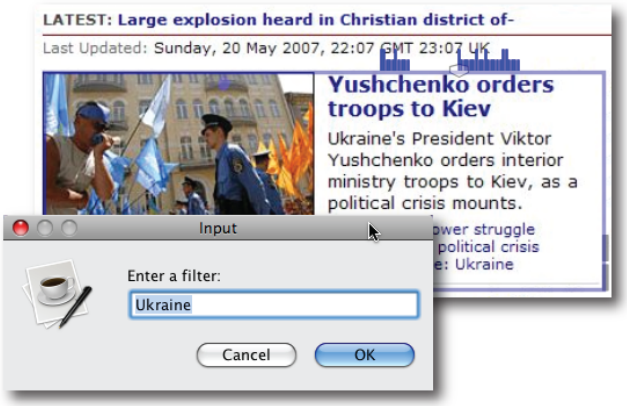


Figure 5: This lens only displays articles that include the word "Ukraine." The scented widget above the slider indicates the location of such articles.

every month between 18:00 and 19:00 (i.e., during 6pm), as shown in Figure 2.

Filtering is visually depicted with a scented widget [28], which is displayed as a small embedded bar graph (Figure 1 and 5). The bar graph is displayed above the slider, indicating the location in time of the matching tuples. As a person moves the slider, the slider snaps to the bars, which act like slider ticks. Note that the bars need not be all of the same height and may reflect different information. A tall bar can indicate the appearance of new content that matches a filter, and a short bar can indicate content that appears previously but still matches the filter. Consider Figure 5 in which a person has specified stories that mention "Ukraine." The tall bars indicate that a new story has emerged about the Ukraine at that time (e.g., "Yushchenko orders troops to Kiev"), whereas the short bar means that the "Ukraine" story still exists on the page but is not new.

### Binding Lenses

People are often interested in multiple parts of a page or parts of multiple pages, as they may be comparing and contrasting different information. For example, in addition to traffic maps from 6pm each day, Ed might also want to look at traffic conditions on days when there are baseball games in the nearby stadium. Zoetrope flexibly allows for the simultaneous use of multiple lenses. Lenses can act independently or be bound together interactively into a synchronized *bind group*. Sliders within a group are linked together, causing them all to move and simultaneously update their corresponding lens.

People may bind lenses for different reasons. For example, to check traffic at 6pm on home game days, Ed can bind a lens for traffic maps at 6pm with a lens for home games from his favorite baseball site. Each lens in a bind group constrains its matching tuples to only include versions allowed by all other lenses in the group. Recall that this is achieved through a trigger filter. Each lens can add a new trigger filter parametrized to the time intervals that are valid according to other members of the bind group. Only tuples that satisfy all trigger filters are allowed. Thus, the resulting stream shows traffic data at 6pm only on days for which there are home baseball games.

Lenses can also be bound disjunctively. For example, one may want to find when book A's price is less than $25 or when book B's price is less than $30 (i.e., one of the two books has dropped in price). Zoetrope supports this type of bind, which is currently obtained by holding the shift key while performing the bind operation. However, this operation creates an interesting twist as it causes data to be *un*-filtered. When binding two lenses in this way, filter operators can be thought of as operating in parallel rather than serially. A tuple passes if it matches *any* filter.

### Stacking Lenses

In addition to binding, Zoetrope also supports the stacking of lenses. For example, consider a person who creates one lens on a weather page, filtering for "clear" weather, and would like to further apply a filter that restricts the selection to between 6 and 7pm daily. Explicitly drawing one lens over the other and then binding them is visually unappealing and does not take advantage of the underlying semantics of the language. Instead, we introduce the notion of lens *stacking*. The toolbar in the Zoetrope window, which allows people to select the type of the lens, can also be used in a specialized binding operation which we call stacking. By dragging a lens selection from this toolbar to the bind button of the lens, a person indicates that they would like to further filter the existing lens. The original lens is replaced, and a new combined lens is generated, which takes the transform and filter from the original selection and augments it with additional transforms and filters. This new lens satisfies both the selection and constraints of the original lens as well as the new one. Furthermore, because some filters and transforms are commutative, stacking provides the opportunity to reorder the internal operations to optimize the processing of tuples.

Finally, we consider the partial stacking of lenses where a person wants to make a sub-selection from an existing lens. For example, a person may apply a textual lens that tracks a specific team in the ranking. The textual lens will track the team no matter where they are in the ranking, but the person would further like to pull out the wins for that team at various time points. Thus, they may create a second structural lens that consumes the selection of the textual lens and selects the wins. While most lenses can be easily stacked without modification, lenses that stack on top of textual lenses require a slight modification to utilize relative information (paths or locations). This subtle modification is necessary because the textual lens selects information that is not in a fixed location in either the $x, y$ space or the DOM tree. Because the textual selection is variable, the structural lens must utilize a path relative to the selection rather than an absolute path.

### Design Considerations

The data Zoetrope manipulates irregularly changes, shifts, and vanishes, and thus our design had to address and accommodate this unpredictable behavior.

*DOM Changes and Selection Failures* Since Zoetrope lenses track information over time, Zoetrope must be able to robustly extract the desired content from different versions of a webpage. However, it is not just the information within pages that changes; occasionally the structure of a page also changes. Indeed, we believe that any page displaying

interesting, dynamic content will eventually undergo a template change which is sufficiently large that most extraction schemes will fail [6]. Others have responded to this challenge by constructing increasingly sophisticated extraction and tracking systems (e.g. [4, 17, 21]). These methods might be readily integrated into Zoetrope, however, by design Zoetrope can partially sidestep this problem. Zoetrope's real-time feedback allows a person to *immediately* see whether a desired selection effect was obtained. This is only possible because Zoetrope works over *historical* data and need not be robust in the face of unknown future changes.

A unique feature of Zoetrope is that we have targeted data that is changing quickly and thus requires crawling at a fairly rapid rate relative to other systems (currently once per hour). It is therefore worth briefly examining the amount of structural change that occurs in this type of data. Using our test collection, we attempted to track the stability of DOM paths. We examined the first crawl of each page and collected all of the leaf DOM nodes. We also looked at a re-crawl from an hour later to remove formatting nodes (e.g., "bold" or "italic") that quickly vanish and are not part of the page's template. The remaining DOM nodes were tested against versions of the webpages a full day, a full week, five weeks, and a nearly a year later. We found that the median survival rate of DOM elements within a page is 98% after one day (87% mean), 95% after one week (83%), 63% after five weeks (61%), and 11% after one year (23%). The differences between the median and mean survival rates indicate outliers in both directions (pages where nearly all elements disappear within a day and pages where nearly all elements remain after a year). Between the initial crawl and the crawl at the end of five weeks, we see that vanished DOM elements, on average, contained 53 characters of text (38% of vanished characters were within links, each link averaging 17 characters). DOM elements that survived, on the other hand, had an average of 131 characters, suggesting greater stability in larger DOM elements. The tremendous amount of path death after a year is unsurprising. It confirms that websites undergo significant template changes that cause extractor failures. Anecdotally, however, we find, as did [6], that although some pages (e.g., *www.bankrate.com* or *money.cnn.com*) undergo major HTML structural changes, the *visual placement* of key features (e.g., a box of mortgage interest rates) is constant even after a year. While we hope that Zoetrope's interaction style allows people to quickly validate their extractions, we also wish to improve our extractors by augmenting DOM paths with visual and textual features.

Extraction failures may occur even when a page is sampled frequently over a short interval. The best response depends on whether such a failure is believed to be permanent or intermittent. Intermittent failures may be caused by randomization of page data (e.g., in a temporary promotion, Amazon interjects a counter of Harry Potter book sales on the front page). The simplest solution for addressing such transient effects is to introduce a dummy (blank) image when an element does not exist. In our experiments, however, we found that introducing blank images for failures results in a disconcerting visual oscillation—the motion appears much like a low frame rate animation. Zoetrope instead displays

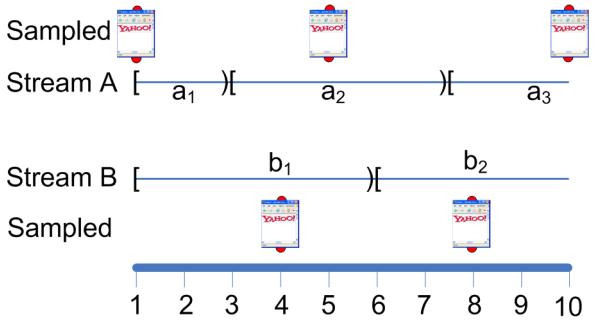


Figure 6: This illustration displays Zoetrope's interpretation of sampled content streams.

a valid extraction from an immediately previous or future version. More formally, suppose that a lens slider is at time $j$ and the nearest sample does not exist. Let $i$ and $k$ denote the times of the closest valid samples such that $i < j < k$. In this case, we display the cropped image from the closer of $i$ or $k$. If time $i$ does not exist (i.e., there are no past instances where the element exists) or $k$ does not exist (no future instances), the first or last valid instance is used respectively. Although artificial, this technique creates the illusion that the selection is continually present over all times. To avoid "deception," a cue (such as a red frame on the lens) can be displayed.

*Windows in Time* It is worth briefly considering the design decision to display past content in the context of a webpage. In our design, the historical information overwrites the part of the page where the lens is located. An alternative would be to respond to slider motion by loading the entire page for a specific time and offsetting that page so that the selected area is underneath the lens (similar to automatically panning the lens "camera" at each step). Although we can enable this mode, we found two main reasons why this type of movement was unappealing.

First, the motion and replacement of the entire page (rather than a small area) was highly distracting. Rather than visually tracking changes in a small area of interest, a person must track changes across the entire screen. Second, it is not clear what should happen when a person creates multiple lenses on the same page. As described above, multiple lenses represent different selections within a page. These selections may not always have the same visual relationship (for example, being 10 pixels apart in one time and 500 pixels apart in another) and may be set to different times. Given these constraints, it may not be possible, without distortion, to offset the underlying page to a position that works for all time periods or combinations of lenses.

*Notions of Time in Zoetrope* When binding lenses on different pages, it becomes important to consider the sampling rate of each webpage, as dynamic pages that change rapidly may be sampled more frequently than those that are more static. Because we wish to present a continuous and easily manipulated abstraction that hides the details of sampling, we make several assumptions: (1) we assume that Zoetrope's sample rate is sufficiently frequent to capture every relevant state of the page, (2) we assume that, when content changes, it does so at the time point which is midway between the times of the two differing samples.


Figure 7: This timeline visualization shows the duration and frequency of news articles on the *cbc.ca* website.

Figure 6 summarizes this interpretation of content streams. Suppose page A is sampled at times 1, 5, and 10; B is sampled twice, at 4 and 8. We interpret this as if stream A has content $a_1$ during the half-open interval $[1,3)$, has content $a_2$ during $[3,7\frac{1}{2})$, and content $a_3$ during $[7\frac{1}{2},now)$. By making these assumptions, we again support the illusion of a fully-defined state over time as a person manipulates a lens over a temporal range.

The details of this interpretation become especially important when a person binds two lenses, making them show the same time (as in Ed's investigation of the effect of baseball games on traffic). Specifically, the interpretation defines a global state for the partially-sampled system, which allows Zoetrope to display corresponding states in multiple lenses simultaneously. In the example of Figure 6, there are four global states, defined by the cross product of the two streams:

$$[1,3)=a_1b_1 \quad [3,6)=a_2b_1 \quad [6,7\tfrac{1}{2})=a_2b_2 \quad [7\tfrac{1}{2},now)=a_3b_2$$

Formally, the evolving global state of the system may be viewed as a database relation where time is a foreign key.

## VISUALIZATIONS

Lenses enable viewing of Web content from different moments in time, but this exploration may be just the first part of satisfying an information need. For example, a book's cost at specific points in time is interesting, but a person may also want to graph the price over time, calculate averages, or test variability. To facilitate this type of analysis, we have created a number of renderers that visualize or otherwise represent selected data. Visualizations, like lenses, create a sequence of transforms, filters, and renderers to display results. Although visualizations can exist independently of a lens, a lens typically defines the data displayed in the visualization. Lenses can thus also be used as a prototyping tool for testing selections and aggregations.

The transforms, filters, and processed streams generated by the lens can be directed to visualization rendering components that implement the visualization itself. For example, in a typical workflow one might place a (potentially filtered) lens on a book price, move the slider to test the selection, and click on the visualization button to graph the price over time. Internally, the visualization step reuses the transform module of the lens and connects it to a time series renderer (see Figure 9). As we describe below, other renderers provide additional visualization alternatives.

### Timelines and Movies

The simplest Zoetrope visualization type is the *timeline* (see Figure 7), which displays extracted images and data linearly on a temporal axis. This visualization allows, for example,

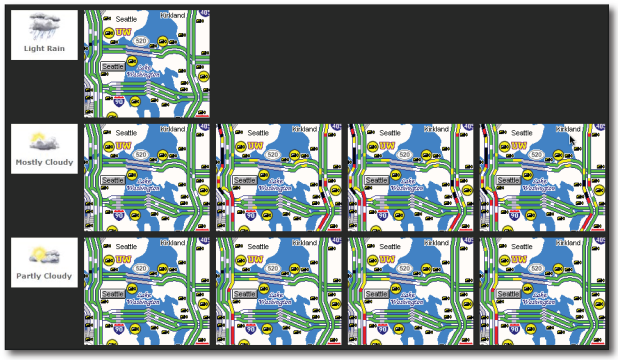


Figure 8: This cluster visualization shows traffic data according to weather conditions.

viewing weather patterns over the course of a year, headline images in stories that mention Iraq, or unique articles about a favorite sports team (all ordered by time). As before, the rendered images visualized in the timeline are live and a person can click on any of the links. Double clicking on any image in the visualization synchronizes the page (or pages) to the same time, allowing a person to see other information that appeared on the page at a particular time. This visualization can also eliminate duplicates and display a line next to each image depicting its duration. This type of display shows when new content appears and how long it stays on a page (e.g., a story in the news, a price at a store) To prevent the timeline from running indefinitely to the right, the visualization can fold the line into a grid with each row denoting activity over a day (or other interval).

The timeline visualization gives an instant sense of everything that has happened over some period. However, other examples are best served by cycling through the cropped images to produce an animated movie. Although this is equivalent to simply pulling the slider through time, a *movie visualization* automates and regulates transitions and looping while the visualization cycles through the images. For example, a static USGS earthquake map can be transformed into an animation of earthquakes over time, helping to pinpoint significant events.

### Clustering

Our timeline visualization is a simple example of a grouping visualization, where extractions are grouped by some variable (time in this case). However, other more complex groupings are also possible. *Clustering visualizations* group the extracted clips using an external variable derived from another stream. For example, a clustering visualization can

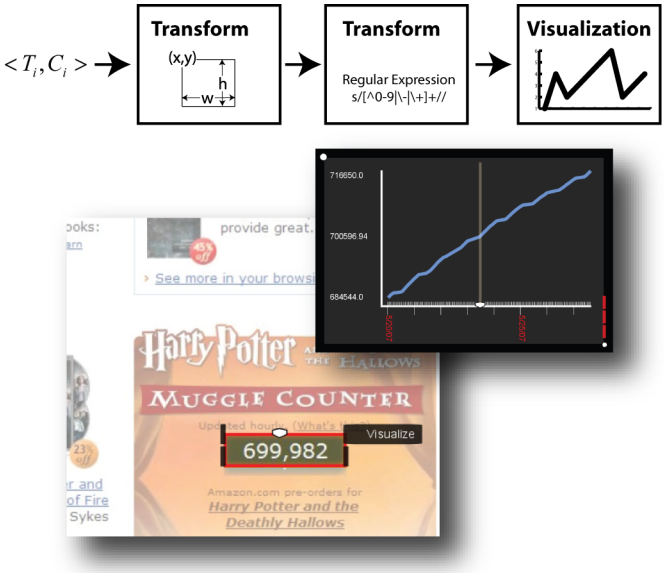


Figure 9: A time series visualization displays the Harry Potter book sales, or Muggle Counter, over time.

merge data from two different lenses, using one lens to specify the grouping criteria while the other lens provides the data. Figure 8, for example, shows a cluster visualization of traffic and weather data. The visualization creates a row for every weather condition (e.g., sunny, clear, rain), and every instance from the traffic selection is placed in the appropriate row depending on the weather condition at the time of the clipping. If it was rainy at 8:15pm, for example, the traffic map from 8:15pm is assigned to the rainy group.

### Time Series

A variety of interesting temporal data is numerical in nature, such as prices, temperatures, sports statistics, and polling numbers. Much of this data is tracked over time; however, in many situations it is difficult or impossible to find one table or chart that includes all values of interest. Zoetrope automatically extracts numerical values from selections of numerical data and visualizes them as a time series. Figure 9 shows a visualization of the number of Harry Potter book sales over time. The slider on the x-axis of the time series visualization is synchronized with the slider in the original lens selection. When the person moves the lens slider, a line moves on the time series visualization (and vice versa).

Time-series visualizations take the output of a lens and transform it to extract numerical values (i.e., $\langle T_i, C_i.Text \rangle \rightarrow \langle T_i, Number_i = \text{FINDNUMBER}(C_i.Text) \rangle$). At present, we use a simple regular expression, but a more complex transformation can be included to handle selections with multiple numbers or mathematical transforms.

### Exporting Temporal Data

Zoetrope offers many useful visualizations, but was also designed for extensibility. Data extracted using Zoetrope lenses is therefore also usable outside of Zoetrope. Systems such as Swivel (*www.swivel.com*) or Many Eyes [27] excel in analysis and social interactions around data, but are not focused on helping people find the data in the first place. Zoetrope is able to generate a temporal, data-centric view of different websites to meet this need. To export data outside of Zoetrope, we created a Google Spreadsheet "visualization" that sends the lens-selected values to the external Google Spreadsheet system. When a person selects this option, appropriate date and time columns are generated along with the content present at that time interval (either strings or numerical data). This scheme greatly expands the capabilities of Zoetrope by allowing people to leverage external visualizations, integrate with Web mashups, or perform more complex analyses.

## IMPLEMENTATION

Zoetrope is a Java-based program implemented using the Piccolo library [1]. Zoetrope primarily relies upon Piccolo for zooming features, but we generally implement our own image functions to manipulate, slice, and display webpages. Zoetrope's implementation is designed to balance the need for interactivity with scale. Pre-rendering and caching certain data structures and images in memory allows Zoetrope to respond in near-real time. Using low-resolution images, the system provides immediate interactivity during slider motion. The system also takes advantage of idle time to background load high-resolution cropped images. Visualizations based upon images also use low-resolution substitutes that are gradually replaced as higher-quality versions become available.

### Crawling

The underlying Zoetrope data is generated by crawling pages at semi-regular intervals. Although at present we have selected a one hour re-crawl rate, this interval could be strategically determined by observing the amount of content change. Initially, the crawler was constructed using a modified *wget* process (the unix command) that retrieved both the page of interest as well as all content attached to a page (JavaScript, CSS, images, flash files, etc.). This, in itself, is generally sufficient to create an approximation of the page as it appeared at a given time. However, dynamically re-rendering pages creates unattractive delays that are inconsistent with Zoetrope's need to support highly-interactive rapid shifts among different times. This led to the previously mentioned optimization of storing pre-rendered images of pages. Image generation was initially achieved with a program containing an integrated Internet Explorer browser. The program, run separately from the crawler, would read the crawled data and produce a rendered image and a modified HTML file that included the rendered location of each element in the DOM. This transformation is consistent with our notion of temporal tuple processing, as the system iterates over page instances and produces a second stream of images. We quickly realized that the difficulty with this approach was that content was frequently missed in the crawl (what *wget* captures is different from what the browser expects), and dynamic content produced a mismatch between what was retrieved and what was rendered (e.g., a JavaScript program that inserts the current time into the page, at rendering time, regardless of when the page was crawled).

We solved this problem with a new system that combines crawling and rendering into a synchronized process. We extensively modified and integrated two Firefox plugins. The first, Screengrab! (*www.screengrab.org*), produces a capture of whatever is being displayed in Firefox. The second, WebPageDump [22], waits until the page DOM stabilizes

in memory and subsequently outputs a single JavaScript-free HTML page, images, and a single CSS document describing all necessary layout constraints. As before, the HTML page is modified to contain the coordinates of all rendered DOM elements. Based upon both of these plugins, our current Firefox-based crawler captures both an accurate representation of the page as it looked at the time of retrieval and sufficient content so that the page can be correctly re-rendered if necessary. Using a Firefox plugin has the further advantage that people can add to their own database as they browse the Web.

Our experimental dataset includes 250 pages and over 1000 samples per page. Each page requires, on average, storing 92Kb (68Kb median) per crawl beyond the original page, which requires an average of 320Kb. This includes all content necessary to render any crawled version of the page (e.g., HTML, all images, Flash files). The first page crawl is much larger, because in subsequent crawls we check linked files, such as CSS and image files, for changes and only download them when they have changed. We compute an MD5 checksum to check for changes. Subsequent crawls are thus only about 28% of the size of the original crawl. Most of the new data, 92%, is due to HTML changes rather than changes in linked files. We believe that our dataset size is artificially high for a number of reasons: (1) the pages in our dataset were selected because they are known to change frequently, (2) we do not currently throw out content that is duplicated between pages from the same website, and (3) we do not compress any files. If we eliminate these inefficiencies, we believe that crawls and storage space requirements could be significantly improved. For example, if we compress the pages using *rzip*, which takes advantage of repetitions across files, the average HTML page crawled 5 weeks after the original visit requires only an additional 2Kb of storage (272 bytes median) over the compressed version of the original page (15% of the size of the original compressed page).

### Backend

The DOM for each version of a page are loaded into an in-memory XML database. Saxon (*www.saxonica.com*), provides XPath query functionality and allows us to rapidly find matching elements or entire versions of pages. An in-memory index also tracks DOM elements by their rendered $x$ and $y$ coordinates. This structure allows us to quickly find which elements are clicked or selected.

## RELATED WORK

Research on the evolution of change on the Web [10] indicates both a high rate of change and large variance in the degree of change. After changing, information on the Web generally vanishes for all intents and purposes. Even archive systems like the Internet Archives (*www.archive.org*) and Google's cache only preserve a partial history. Previous copies, such as they are, are not generally accessible in a useful way, but instead require a person to manually search for and individually collect each page.

In response to the need for temporal data, a number of solutions have emerged in the research community. Arguably one of the first, the AT&T Difference Engine [9] archived pages and displayed differences between consecutive versions of pages. Because HTML is semi-structured, researchers in the database and data mining communities have expanded ideas on XML differencing to HTML/Web differences (extensively surveyed in [12]). Zoetrope is not explicitly designed for comparing two page versions to each other (though this can certainly be implemented). Rather, we are interested in *many* instances of a given page.

Another related area of research includes Web-clipping systems that allow people to monitor webpage regions of interest [13, 24, 26]. These systems are typically focused on tracking future changes rather than working with previous versions of webpages. These efforts have also inspired commercial efforts, such as Internet Explorer 8's *WebSlices*. Multi-page clipping and mashup/extraction applications (e.g., [7, 8, 14, 15]) are valuable tools for extracting and combining data from the present Web, but are not designed for historical versions. Of notable exception are programming by demonstration (PBD) systems such as Koala [17], which must take into account the changing Web to preserve the validity of macros and scripts going forward. Zoetrope implements Web-clipping features in a number of visualizations (time lines, clusters, etc). Our architecture is well-suited to such applications and we hope to investigate them further in the future.

A second major area of research has been the visual display of temporal differences in information. Certain solutions are targeted at news while others act more generally to represent differences between versions of webpages. These include techniques for converting textual information into time series [11], spatio-temporal visualizations [23], and other difference visualizations (e.g., [18, 19]).

Architecturally, a number of database visualization systems are related to the stream and operator design in Zoetrope. Particularly, DEVise [20] was originally constructed to operate on streams of data that required visualization. Similarly, Polaris [25] operates to visualize relational data. In both, the operators provided by the system are primarily targeted at the rendering step (deciding how graphs, charts, and other visualizations should be constructed). Zoetrope instead focuses on unstructured temporal Web content from which data can be extracted for these types of visualizations. Other systems-focused research that could support future additions to Zoetrope include work on automation and scalability issues of large-scale Web archiving (e.g., [5]) and full-text search (e.g., [2]).

Finally, although these approaches have not been targeted at temporal Web data, we draw inspiration from the Video Cube [16] and Magic Lenses work [3]. The former allows video stream frames to be layered and "sliced" to find an abstraction of the video. In the latter, a magic lens is a widget that can be placed directly on a document to illuminate the underlying representation while maintaining the visual context of the document.

## CONCLUSIONS

This paper presents Zoetrope, a system for interacting with the ephemeral Web. Zoetrope allows people to interactively access and manipulate a historical record of the evolving Web. Familiar pages, which have previously

existed only in the present, now have the added dimension of time. To help manage and utilize this multi-version Web, Zoetrope implements a number of interactions based in lenses and visualizations that allow people to access and analyze historical information through contextualized selections and extractions. Furthermore, developers can use Zoetrope to rapidly prototype and test temporal queries through Zoetrope's visual query interface. With the Zoetrope system, we introduce a novel semantics for dealing with temporal Web data and show a small number of operators that can be utilized for many different tasks. Our present implementation of Zoetrope has only scratched the surface of potential lenses and visualizations. By creating an underlying language for selecting and processing temporal page data, we hope to inspire and enable others to explore new interactions and applications in this domain.

In addition to continuing to build new visualizations and extractors, we are extending the semantics of the underlying language to support more sophisticated queries and binding operations for specifying temporal logic operations and functions (e.g., the gas price for two days *after* oil prices are $60 per barrel). Also, the highly regular sampling of our dataset allowed for a number of assumptions in our semantics, and we hope to further explore irregular crawl rates and interpolating data as necessary.

## ACKNOWLEDGMENTS

We would like to thank our reviewers for their helpful comments and advice. Additional thanks to Jaime Teevan, Susan Dumais, Rob Miller, Adam Weld, and the GRAIL group for their feedback on this work. This work is partially supported by the WRF / TJ Cable Professorship. Eytan Adar is supported by an NSF Graduate Fellowship and by ARCS.

## REFERENCES

1. Bederson, B. B., Grosjean, J., and Meyer, J. (2004). Toolkit Design for Interactive Structured Graphics. *IEEE Trans. Softw. Eng.*, 30(8), 535-546.
2. Berberich, K., Bedathur, S., Neumann, T., and Weikum, G. (2007). A Time Machine for Text Search. *SIGIR '07*. 519-526.
3. Bier, E. A., Stone, M. C., Pier, K., Buxton, W., and DeRose, T. D. (1993). Toolglass and Magic Lenses: The See-Through Interface. *SIGGRAPH '93*. 73-80.
4. Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R. C. (2005). Automation and Customization of Rendered Web Pages. *UIST '05*. 163-172.
5. Boyapati, V., Chevrier, K., Finkel, A., Glance, N., Pierce, T., Stockton, R., and Whitmer, C. (2002). ChangeDetector: A Site-Level Monitoring Tool for the WWW. *WWW '02*. 570-579.
6. Dontcheva, M., Drucker, S. M., Salesin, D., and Cohen, M. F. (2007). Changes in Webpage Structure over Time. *Technical Report TR2007-04-02, UW CSE*.
7. Dontcheva, M., Drucker, S. M., Salesin, D., and Cohen, M. F. (2007). Relations, Cards, and Search Templates: User-Guided Web Data Integration and Layout. *UIST '07*. 61-70.
8. Dontcheva, M., Drucker, S. M., Wade, G., Salesin, D., and Cohen, M. F. (2006). Summarizing Personal Web Browsing Sessions. *UIST '06*. 115-124.
9. Douglis, F., Ball, T., farn Chen, Y., and Koutsofios, E. (1998). The AT&T Internet Difference Engine: Tracking and Viewing Changes on the Web. *World Wide Web*, 1(1), 27-44.
10. Fetterly, D., Manasse, M., Najork, M., and Wiener, J. (2003). A Large-Scale Study of the Evolution of Web Pages. *WWW '03*. 669-678.
11. Fitzpatrick, J. A., Reffell, J., and Aydelott, M. (2003). BreakingStory: Visualizing Change in Online News. *CHI '03*. 900-901.
12. Grandi, F. Introducing an Annotated Bibliography on Temporal and Evolution Aspects in the World Wide Web. (2004). *SIGMOD Rec.*, 33(2), 84-86.
13. Greenberg, S. and Boyle, M. (2006). Generating Custom Notification Histories by Tracking Visual Differences Between Web Page Visits. *GI '06*. 227-234.
14. Hartmann, B., Wu, L., Collins, K., and Klemmer, S. R. (2007). Programming by a Sample: Rapidly Creating Web Applications with d.mix. *UIST '07*. 241-250.
15. Huynh, D., Mazzocchi, S., and Karger, D. (2005). Piggy Bank: Experience the Semantic Web Inside Your Web Browser. *ISWC '05*. 413-430.
16. Klein, A. W., Sloan, P.-P. J., Finkelstein, A., and Cohen, M. F. (2002). Stylized Video Cubes. *SCA '02*. 15-22.
17. Little, G., Lau, T. A., Cypher, A., Lin, J., Haber, E. M., and Kandogan, E. (2007). Koala: Capture, Share, Automate, Personalize Business Processes on the Web. *CHI '07*. 943-946.
18. Liu, B., Zhao, K., and Yi, L. (2002). Visualizing Web Site Comparisons. *WWW '02*. 693-703.
19. Liu, L., Pu, C., and Tang, W. (2000). WebCQ-Detecting and Delivering Information Changes on the Web. *CIKM '00*. 512-519.
20. Livny, M., Ramakrishnan, R., Beyer, K., Chen, G., Donjerkovic, D., Lawande, S., Myllymaki, J., and Wenger, K. (1997). DEVise: Integrated Querying and Visual Exploration of Large Datasets. *SIGMOD Rec.*, 26(2), 301-312.
21. Phelps, T. A. and Wilensky, R. (2000). Robust Hyperlinks and Locations. *D-Lib Magazine*, 6(7/8).
22. Pollak, B. and Gatterbauer, W. Creating Permanent Test Collections of Web Pages for Information Extraction Research. *SOFSEM '07*. 103-115.
23. Rennison, E. (1994). Galaxy of News: An Approach to Visualizing and Understanding Expansive News Landscapes. *UIST '94*. 3-12.
24. schraefel, m.c., Zhu, Y., Modjeska, D., Wigdor, D., and Zhao, S. (2002). Hunter Gatherer: Interaction Support for the Creation and Management of Within-Web-Page Collections. *WWW '02*. 172-181.
25. Stolte, C., Tang, D., and Hanrahan, P. (2002). Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Trans. on Vis. and Comp. Graphics*, 8(1), 52-65.
26. Sugiura, A. and Koseki, Y. (1998). Internet Scrapbook: Automating Web Browsing Tasks by Demonstration. *UIST '98*. 9-18.
27. Viegas, F. B., Wattenberg, M., van Ham, F., Kriss, J., and McKeon, M. (2007). ManyEyes: A Site for Visualization at Internet Scale. *IEEE Trans. on Vis. and Comp. Graphics*, 13(6), 1121-1128.
28. Willett, W., Heer, J., and Agrawala, M. Scented Widgets: Improving Navigation Cues with Embedded Visualizations. *IEEE Trans. on Vis. and Comp. Graphics*, 13(6), 1129-1136.