Generating Custom Notification Histories by Tracking Visual Differences between Web Page Visits

Saul Greenberg and Michael Boyle

Department of Computer Science, University of Calgary
2500 University Drive N.W.
Calgary, Alberta, Canada T2N 1N4
E-mail: {saul, boylem}@cpsc.ucalgary.ca

ABSTRACT

We contribute a method that lets people create a visual history of custom notifications to track personally meaningful changes to web pages. Notifications are assembled as a collage of regions extracted from the fully rendered (bitmap) representation of the web pages. They are triggered when visual changes between successive visits are detected within regions. To use the system, a person specifies a notification by clipping personally interesting regions from the bitmap representation of a web page and reformatting them into a small collage. The person then specifies regions on the page that will be monitored and compared for visual differences over time. Based on this specification, the system periodically revisits the page in the background on behalf of the user and automatically generations a notification (the collage plus a title and timestamp) when differences are detected. Finally, the person views the generated notifications in several ways: as only the most recently changed version (to illustrate current state), or as an image history that can be individually browsed or played back as a continuous video stream.

CR Categories: H.3.3 [Information Search and Retrieval]: Information filtering, selection process, H.3.4 [Systems and Software]: Distributed systems, H.4.3 [Communications Applications]: Information browsers.

Keywords: Notifications, information customization.

1 INTRODUCTION

People often revisit web pages to monitor personally interesting and dynamically changing information. The actual information can be quite varied, with typical examples including stock prices, currency values, weather reports, traffic conditions, on-line status of people, news postings, sports reports, flight status, upcoming meeting appointments, webcam snapshots, blog entries, new academic publications, and so on.

Yet revisiting pages can be tedious. Consequently, we are now seeing a proliferation of *notification systems* that let people selectively subscribe to personally interesting and dynamically changing information sources offered by a content provider. Manual polling is replaced by a system that delivers relevant and timely changes of that information – notifications – to the people interested in it (e.g., Alerts [9], RSS [12], Sideshow [2], various research systems [3,4,6,8,15]).

The problem is that content providers offer only a few varieties of notifications, and these often come with strings attached (fees, advertisements, account creation, restrictions, prescribed delivery timings, inappropriate information detail, etc). When a person's interests do not match the offerings available, they are essentially excluded from the service. This mismatch will not disappear over time because most content providers are motivated by commerce and mass market needs vs. individual needs and idiosyncrasies.

Several systems – mostly research ones – do allow people to create their own custom notification elements [3,4,6,8,15]. Unfortunately, creation is restricted to technically savvy programmers who know how to tap into, parse, manipulate and display web services or other raw information sources. This is clearly not an option for the vast majority of people.

Consequently, our research goal was to develop a method that would let non-programmers generate their own custom notification elements from information found on web pages. We wanted these people to be able to eas

Figure 1: The annotated Notification Viewer, showing an example of thirteen numbered notifications

information chunks are associated with specific spatial regions on a page;

- depending on the page, the approximate position of a spatial region across updates can be defined from its absolute position on the page, or its relative position from the top or bottom of the page, or its relative position from a constant visual image landmark;
- these positions may not always remain precisely the same between page updates, but are close enough that a search within a small area surrounding a region can be used to verify that image changes detected across updates are a result of real changes vs. a small shift in image location;
- updates to that information can be detected and extracted by monitoring that region for image changes;
- because the rendering process may introduce small visual artifacts, e.g., text anti-aliasing, the change detection algorithm must include a threshold where small image changes between pages are considered equivalent;
- updated pages may repeat information over a series of changes, e.g., when a news provider randomly chooses one of (say) 5 candidate 'articles of the day' to display in a region on successive page visits; thus the history of changes should be searched to filter out duplicates.

While this sounds like a complex set of interacting conditions, we believe that people can use their knowledge of layout conventions, visual cues, content genre, and experiences visiting a page over time to intuitively decide (or later debug) how spatial constancy on a page is maintained across updates.

3 What the User Sees and Does

We begin with the end user experience. We walk through a detailed example of how 'Jane' creates a custom notification from a source web page using the *Notification Editor* (annotated in Fig. 2), which eventually appears at the top left of the *Notification Viewer* (annotated in Fig. 1). Some technical explanation accompanies this description. We stress that describing this interactive process through text makes the system appear more complex than it actually is; in practice, the action sequence outlined below is easily accomplished in under a minute. As well, portions of these steps are optional; defaults for most settings usually suffice.

3.1 Creating the Custom Notification

3.1.1 Motivation

Jane, a flextime worker and an avid skier, sets aside work for a few hours to ski at a local ski area when conditions, crowds and weather prove enticing. While the ski area web site offers several webcam images of current conditions, it is tedious to revisit, and the page is too large to keep on constant display for manual refreshing. As well, the static webcam images do not give her a sense of how conditions are changing over time. Consequently, she composes a custom notification to help her monitor two webcams. Fig. 1, top left shows the end result (notification #1).

Figure 2. The Notification Editor, annotating and illustrating how a notification is created from a web page

3.1.2 Selecting the information source

Jane types in the Sunshine Village ski area's URL in the Address Bar. The current Sunshine page appears as a bitmap in the *Content Editor* (left pane) of the Notification Editor (Fig. 2).

3.1.3 Specifying a landmark

Subsequent page visits may insert variable-sized advertising banners in the page, which can shift the information of interest around the page. To remedy this, Jane specifies one or more visual landmarks near the information of interest to her; other image operations are then performed relative to these landmarks. Selecting the *Landmark* tool from the *Tool palette*, she drags a small region atop a portion of the 'LIVE WEBCAMS' text (Fig. 2, top left). The system will later search revisited versions of this page for this landmark image (looking at the surrounding area first) in order to calculate its new position if it has shifted.

3.1.4 Specifying change region(s) of interest

Jane wants to see the most current webcam image. Yet she does not know how often these images are taken, nor does she want to have to deal with 'frozen' images that result from the camera being turned off. She does notice that the main webcam image is accompanied by a title bar showing the time it was taken: if this time changes, this suffices to warrant a new notification. Consequently, she uses the *Change region* tool to drag out a region atop the titlebar snapshot time (Fig. 2, top left), and indicates in the *Region Properties* (Fig. 2, top right) that this region's location is relative to the landmark she had previously created. The system will later monitor this region for changes as determined by visual differencing between page revisits.

If she wishes, she can fine-tune the properties of this region through the *Region properties* pane (top right, Fig. 2). In particular, using the *Change trackbar*, she can specify a change threshold, i.e., the amount of visual changes that must occur in this region between the current page and its previously visited version if it is to trigger notification generation. To fine tune this

threshold, she can use the *Test change regions* tool to display before, after, and 'differenced' images of the regions between successive page visits (not shown). In this case, she leaves it at its smallest setting, as she believes that any visual change in this region should result in the posting of a new notification.

Jane can continue this process by creating other change regions of interest on this page. If multiple change regions are specified, each may be assigned its own landmark or change threshold: a notification is triggered for the whole page if any one of the change regions differs across page visits. In this example, the single change region suffices.

3.1.5 Specifying the notification history and revisit frequency

Using the controls in the *Notification properties* pane (mid-right, Fig. 2), Jane leaves the *Record history* checkbox activated. This instructs the system to save a history of all generated notifications. She can review all generated notifications via the player controls attached to each notification (Fig. 1).

Also, Jane adjusts the trackbar in that pane to specify how often the web page should be checked for changes. The system will automatically revisit the page based on this interval and will compare the bitmaps defined within its change regions to those on the previous version of the page. This time interval can range from 15 seconds, to minutes, hours, days, and even weeks. Because she wants moment by moment ski hill conditions, she sets the revisit frequency to 1 minute.

3.1.6 Assembling a notification

When the system generates a notification (because it detected changes) it creates and publishes by default a scaled down version of the whole web page as a high quality thumbnail. However, Jane would rather see the two webcam images as a picture in picture, one showing her the base of the hill, and the other showing lineups at a popular lift as illustrated in notification #1 in Fig. 1.

Jane selects the *Copy region* tool and drags out rectangles over regions on the web page to appear in the notification. These copy regions also appear in the *Image editor* pane (lower right, Fig. 2). In this pane, Jane can at resize, reposition, and restack the image fragments that come from copy regions to compose a custom notification: this visual layout is what will be presented to her when a change to the page is detected. A copy region can also be associated with a landmark.

As seen in Fig. 2 and shown in its generated form in Fig. 1, Jane will create a notification captured from four copy regions: the Sunshine logo (upper left, copy region 3), the time of the webcam snapshot (top 1st webcam image, copy region 1), the featured webcam image (middle, copy region 2), and a 2nd webcam image (bottom, copy region 4). Full sized versions of these are automatically copied to the Image Editor – arrows in Fig. 2 show the source copy regions in the Content editor, and where they appear in the Image editor. She creates a visual banner by resizing and abutting the logo and time regions. She moves the main webcam image underneath, and then resizes and moves the 2nd webcam image to create the desired picture in picture effect.

At this point, Jane has completely specified the notification description: the information source (the web page), the regions of interest on it, how often that page should be revisited to see if content in those regions has changed, and the visual appearance of the notification to be generated when changes are detected. She can now save and later reload this specification (stored as an XML file) for further editing. She can email this file to friends so that others can use or further customize this notification to meet their own needs.

3.1.7 Generating notifications

After specifying the URL location of a notification server through the *Connect* tool, she activates the notification *Generator* . From now on, the notification generator will use the notification description to determine if it should generate a notification: details are described in Section 4. For immediate local feedback and further fine-tuning, the Content editor pane and the Image editor pane are updated to display the most recently visited version of the page and the corresponding generated notification. Of course, other notifications generated from different pages may be published to the server at the same time.

3.2 Viewing Notifications

Later, Jane views this and other notifications by starting a Notification viewer client, which connects to the notification server. She immediately sees several notification items each in their own window (Fig. 1): these all represent items she had previously specified and that have visually changed over the course of time. As annotated in Fig. 1, all notifications contain: a title, a clickable *Web link* URL to display the web page in a browser, the *time* of the posting, the number of *frames* in a notification's history, and *player controls* to navigate a notification's history. Jane can manipulate notifications in several ways. If she resizes one, the image is scaled to fit. Or she can close a notification after reading it: it reappears when a new notification is generated.

Jane sees the Sunshine notification at the top left of Fig.1, and notices that it has 201 frames in its history available for viewing. Using the *player controls*, she replays the entire history as video sequence: several hours are condensed into a minute (rate is adjusted by the *playback speed* trackbar, Fig 1. bottom). She sees a spectacular sunrise as dawn breaks over the Canadian Rockies, workers completing their grooming of the slopes, the buzz of activity as crowds arrive which gradually surges into a morning rush, the brilliant sunshine replaced by darkening clouds as a storm arrives, and crowds leaving early because of worsening conditions. While she was thinking of taking the afternoon off to go skiing, she decides to wait for a better day.

Jane had previously created other notifications using techniques similar to the above. To show the range of possibilities, and as numbered in Fig. 1, these include:

- *feature news stories* from several news sites (CNN #7, BBC #8, CBC #10),
- *featured articles* from technical magazines (Slashdot #11, PC Magazine, #12),
- *traffic webcams* (Calgary traffic #5),
- *scenery webcams* (Banff townsite webcam #2),
- *weather reports* (Calgary weather #4),
- *daily comic strips* (PhD comics #3, Dilbert #6),
- *updates to publications* (Grouplab Publications #9),
- periodically updated photos (Wagar #13),
- *blog entries* (not shown),
- *changes to her group's schedule* (not shown)

For several notifications, such as the Slashdot and news web sites, she sees that several updates have been captured in their histories. For the ones of interest, she uses the player controls to sequentially step through the previous articles.

Over the course of the day, these notifications are updated and new ones appear as information is published to the notification server. In this way, Jane quickly sees new postings as they arrive. If she steps away from her machine, she can catch up on any

missed notifications when she returns by replaying the individual notification histories.

4 IMPLEMENTATION

The architecture for this system is reasonably straightforward (Fig. 3), but is described here in sufficient detail to encourage replication.

Before delving into this detail, we should mention that our idea of using image clipping as the basis for creating custom notifications is inspired by Tan et. al.'s WinCuts [14] and Fujima et. al.'s C3W [5]. While neither are notification systems, both use the idea of image clipping to allow users to customize an information display. With WinCuts [14], users can clip regions of live source windows and assemble them to create a new interactive display, which can even appear on a different machine. That is, WinCuts allows 'remixing' of an existing user interface. Similarly, C3W – Clip, Connect and Clone for the Web – lets users create new web interfaces (especially forms) by remixing regions garnered from various web pages [5].

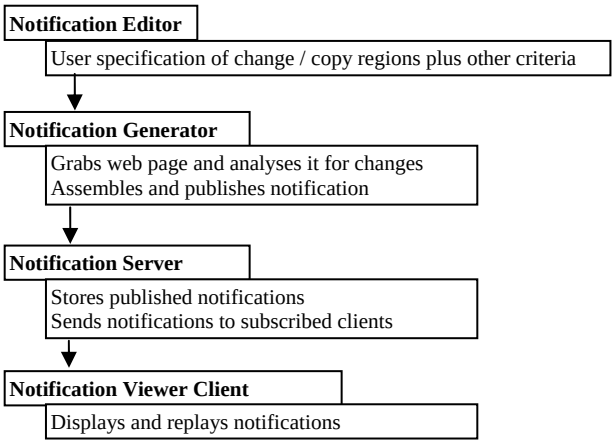

Figure 3: Basic architectural components

4.1 The Notification Editor

The *Notification Editor*, seen in Fig. 1 and abstracted in Fig. 3, provides a graphical interface that allows the user to specify the information necessary to track web pages for changes and to build a custom notification. This specification, which can be saved as an XML file and later reloaded, includes: the *Url* of the page to be monitored, the *Monitoring frequency* describing how often that page should be checked for changes, a list of *Change regions* that specify the rectangular regions on the page to check for bitmap differences according to a specified *Threshold*, a textual *Title* of the notification to be generated, a *History length* value that indicates the length of the history list of notifications to be stored, a list of *Copy regions* that indicate what regions on the *Source web page* should be copied into the *Destination notification* image, as well as the location and (scaled) size of those copies on the destination. A list of *Landmark regions* are also stored and associated with particular *Change* or *Copy* regions.

Aside from its user interface, the only special features required by the Notification editor are the ability to capture and render a web page into a fixed-size off-screen bitmap (to limit differences caused by dynamic reformatting of page content to fit different window sizes) and perform various image capture and manipulation functions (e.g., cropping and scaling). To simplify this task, we used the Collabrary Toolkit [1], which provides an API to all these required features.

4.2 The Notification Generator

The *Notification Generator* uses the specification produced by the Notification editor (Fig. 3). It begins by rendering the web page off-screen at the specified monitoring frequency.

If one or more landmarks had been specified, it then tries to find them on the page. It does this by going to the original location of the landmark and checking to see if the underlying image region on this page matches the original landmark image. If it doesn't, it then searches the area around that landmark to see if an exact image match can be found. That is, if there is no match at the landmark origin (0,0), it iterate through other possible origins ranging from (0,0) up to +/- (n,n), where n defines how far it should look from the original landmark origin before giving up. In the current implementation, n=200. Because landscape images are usually small, this search can be done fairly quickly.

If the landmark is found, it will then use the landmark's actual location on the current page to calculate the positions (as an offset) of associated Change or Copy regions. It compares the monitored Change regions of the current page with a stored version of the last one it has seen. If a change is detected that surpasses the region's Change threshold, the Notification Generator builds a notification comprising the notification title, the URL, and a timestamp. It then creates the composite notification image by locating, capturing, and scaling the Copy regions from the source page into the destination notification.

4.3 The Notification Server.

The generator publishes this information to a *Notification Server* [1] (Fig. 3), which in turn stores and publishes the notifications to various clients. The published data is simple. A unique ID identifies the notification. The URL, title, timestamp and composite image are the notification contents. If the *History length* is 1, then the new timestamp and image data over-writes the previously published contents. If it is greater than 1 then each notification is added to a history list maintained by the notification server. If the number of notifications exceeds the predefined history length, then the oldest notification is removed before a new one is inserted.

4.4 The Notification Viewer Client

The *Notification Viewer Client* subscribes to the notification server, and displays new and changed notifications to the user as they arrive. The client handles all notifications in the same way, regardless of where they were produced. It only needs to know how to display a textual title, timestamp, URL, and a single image. The only special capability we insist on is the ability to play back the notification history frame by frame or as a 'video'. To do this, our client simply navigates the notification history held by the notification server, where it retrieves and displays that frame's timestamp and image.

While our viewer client interface presents individual notifications as a collage of possibly overlapping items positioned in a large window (Fig. 1) [6,3], other quite different interfaces could have been constructed instead. For example, we also implemented a viewer as a media item within the Community Bar [8], a real time groupware tool that posts common notifications to an online group. The Community Bar uses the Sideshow metaphor [2] to let people monitor peripheral information, and to drill down into information as desired. In our version, all new notifications are scaled to appear within a single media item. Clicking on its 'tooltip grande' allows a person to browse a larger version of the current notification and its history through its player controls. Finally, a person can raise a separate third window similar to the one shown in Fig. 1 to show all notifications. Similarly, it would

be straightforward to send notifications to PDAs or cellphones, or RSS [12].

4.5 Limitations of the Current Version

Our actual implementation differs somewhat from the description above, in that it combines the Notification Editor and the Notification Generator into a single process. While this suffices to for a proof of concept system, it is not ideal for day to day use. The problem is that a user has to open an Editor/Generator for every web page they wish to monitor; if a previous specification for a page exists, they must still load it into an Editor instance before the Generator can be activated. For example, the 13 notifications seen in Figure 1 imply that the end user also has 13 active Editor/Generators running. This introduces unnecessary clutter and complexity in the interface, and unneeded overhead to re-establish notifications between logins.

We stress that this is not a limitation of the ideas presented in this paper, but rather an artifact of our current implementation. Indeed, we discuss it here only because we include a link to our program so others can try it (see Software Availability at the end of this paper). A better design, which would be straight-forward to program, would be to separate the Editor and the Generator into two different programs. The end user would use an Editor instance only as they create or modify one or more specifications; the Editor would store these specifications in a convenient location (perhaps on a server). A single Generator process would then read in all the stored specifications, check the indicated pages at the indicated monitoring frequency, and generate notifications.

5 Discussion

5.1 Images vs. HTML?

Our method shares a common goal with Sugiura et. al's Internet Scrapbook [13], which is to let people specify and collect information fragments from web pages for later viewing. However, Sugiura uses an HTML pattern matching and machine learning methodology that is completely different from our image-comparison approach. (Their presentation of these fragments into a single web page with no playback capability also differs from our notification generation and playback approach). This paper does not attempt to evaluate whether HTML vs. images is 'better'. Rather, each has their own strengths and tradeoffs (as partially described in the next section), and ultimately these will be used to decide what method is appropriate.

The choice of whether to use images vs. HTML methods is reminiscent of how shared window systems intercept screen fragments: while early systems typically intercepted low level graphics protocol, later systems advocated bitmap capturing instead [11]. The debate of which is better for some situations is ongoing.

5.2 Power

The power of our image-based visual differencing method is its simplicity. First, it leverages the *surface representation* strategy of Olsen et al., [10] who argue that the major strength of surface representations (i.e., the visual images people see) is that they are human consumable. That is, in contrast to the abstract data that produces the visual (such as HTML extraction [13]), surface representations are designed to be understood by the user. For change regions, users intuitively know from past experiences and from web page genres what parts of a web page will change over time. For creating a notification, people simply harvest the visual information that is most interesting to them, exploiting image copying, pasting and resizing skills already learned through the many image and photo editors they have used. Unlike the HTML approach which can only work on selections that follow a well-formed HTML fragment [13], notification images generated from partial, overlapping and remixed HTML fragments are easy to do.

Second, our strategy works with a broad set of arbitrary data sources because it leverages bitmap renderings of the information source as a lowest common denominator for information representation and extraction. Again, unlike the HTML-specific method [13], our approach only requires a standard web browser object to render the page to a window. Other than that, notifications generated from our image-based method needs no knowledge of the language that generated it, nor of other Web protocols that may evolve over time.

Third, because the notification data generated comprises only a few text fields and a list of images, it is easy to collect and distribute. Our method could be easily implemented atop any notification service, including an SQL database.

Fourth, notification viewers only need to know how to render received images and a few simple textual attributes. They do not need to know (or care) about the original source data used to generate that image. As well, image-based notifications are easily displayed on devices that do not know how to render complex HTML, e.g., photo cell phones.

Fifth, images as the medium for representing and presenting a notification affords special power. It is easy to keep and exploit visual history lists of notifications. Because each notification in the history is just a visual, this medium lends itself to frame by frame browsing of notification histories as well as high-speed playback as a video stream.

Sixth, the cut-and-paste approach to composing notifications as a visual collage means that users can exclude regions which contain information of little interest, especially advertisements, blinking icons, scrolling marquees, white space, etc. Resulting notifications are compact. For example, compare the source web page of Fig. 2 (which must be scrolled if one is to see all its contents), to the extracted notification in Fig. 1. We should mention that while we have not implemented a means to create a single notification by cutting and pasting from several source web pages (as in the Internet Scrapbook [13]) this could be easily added to our system, albeit at the cost of interface complexity.

Finally, the user experience – while complex to describe in text – is lightweight and fun. Creating notifications in the Notification editor is easy to do because it relies only on what the user sees and understands visually. The automatic page revisitation and use of visual differences to filter out personally irrelevant or uninteresting changes eliminate the tedium of having to manually poll for changes to information. Information sources come alive in the Notification viewer as changes are tracked and notifications generated as they unfold over the course of a day. The ability to play back the history of a web site as a browsable sequence or video stream adds information and liveliness not normally discernable in a web page's static images.

5.3 Limitations

The image-based approach to creating a notification relies on a web page retaining its basic spatial layout between visits. We believe (but have not verified) that this is especially true for information near the top of the page. This is obviously a direction of future research. Still, others have empirically tested this notion of layout consistency, where they found that information extraction using spatial location succeeded over 90% of the time [13]. The problem is that this information does shift around somewhat, especially due to resized banner advertisements that typically appear at the top or side of a page. Our use of landmarks remedies this problem for the majority of cases. Our own experiences suggest that landmarks combined with change and copy regions often created the notification we wanted. For the odd

time it did not, minor fine-tuning usually sufficed to correct the problem.

In spite of our optimism, there are classes of (sometimes solvable) 'problem' pages.

5.3.1 Radically changing page layouts

There will always be some pages where changes result in completely different page layouts. In these cases, a person can revert to the (default) strategy of having the system capture the whole page as a thumbnail and to use that as the notification. The thumbnail is captured at a just readable size, so people can resize the image if they wish to read what is 'above the fold'.

5.3.2 Minor edits

Pages may be modified by minor edits, e.g., the addition of a single character or line break. This can shift all text in the image, thus changing the image significantly even though the semantic content has not altered that much. This results in a false positive notification.

5.3.3 Unpredictable page completion times

Many embed programs (e.g., Java applets and Flash animations) use client-side scripting to finalize document construction after the page is loaded. On these types of pages, grabbing an image after it generates a 'document complete' event can result in a web page with missing elements. To avoid this problem, our heuristic was to delay capturing a page's bitmap until several seconds after the 'document complete' event. This usually sufficed to let these extra page elements load correctly.

5.3.4 Animations

Our method works on static images, not animations. To overcome this, our capture processes mutes animations generated by (say) dynamic HTML and scripting. While this works well for pages where the primary areas of interest is static, it would not work if the person wanted to capture a region of interest that was animated.

5.3.5 Inaccessible web pages

Some web pages cannot be navigated to directly: password protected web sites, server redirection, popup browser windows, CAPTCHA challenge/responses, and so on. It should be possible to use programming by example techniques to automate past these extra actions.

5.3.6 Errors

Errors are a fact of life, due to (say) client-side script errors, server-side web application errors, network connectivity issues, and so forth. These generate their own web pages that ultimately lead to 'spurious' notifications. Although not implemented in our system, error detection and filtering strategies can help minimize these spurious notifications.

5.3.7 Copyright protection

Finally, there are the legal and ethical issues involved when web information is clipped without permission. While many consumer advocates believe that falls under 'fair use', there is a large lobby effort by corporations to make remixing of web material illegal [7]. Other concerns arise from the capture of static information (e.g., webcam snapshots) as a replayable video, for the information revealed may not be what the author intended.

6 CONCLUSIONS

At the high level, we contribute a novel method that uses visual differences in user-selected regions of a web page's surface representation – a bitmap of the fully rendered page – to serve as the basis for a notification system: custom notifications are themselves harvested and assembled from user-specified regions of that bitmap. While aspects of this resemble the clipping approach for custom interface design advocated by Tan et. al. [14] and Fujima et. al. [5], its application to notification generation gives a very novel and completely different effect. We also contribute:

- how custom presentations of notifications can be assembled from changes to these images (leveraging ideas found in [13,14,5]);
- how the underlying algorithms as well as the notification server only needs to handle very rudimentary data types (e.g., images and meta-information such as the source URL for that image): this implies that this method is very generalizable, and can be used across many different notification engines;
- how the presentation of image notifications also needs only rudimentary knowledge of the data source, i.e., how the notification viewer renders and/or scales images; and,
- how notifications can be collected as an image history, which supports rich playback mechanisms that can be used to present not only the current state of that information, but how it has changed over time.

Future work includes empirically verifying our assumptions about spatial regularities, evaluating how users exploit these regularities in a deployed system, and studying limits to user tolerance of notification errors

SOFTWARE AVAILABILITY

An executable version of the software described in this paper can be found by going to http://grouplab.cpsc.ucalgary.ca/cookbook/ and following the links to Web Tracker.

ACKNOWLEDGEMENTS

This research was partially funded by the NECTAR NSERC Research Networks grant, and its industrial partners Smart Technologies and Microsoft Inc. We also thank James Eagan from Georgia Institute of Technology: his discussions of end-user creation of notification inspired our interests into this area.

REFERERNCES

[1] Boyle, M. and Greenberg, S. Rapidly Prototyping Multimedia Groupware. *Proc Distributed Multimedia Systems* (DMS'05), 2005.

[2] Cadiz, J., Venolia, G., Jancke, G., Gupta, A. Designing and deploying an information awareness interface. *Proc ACM CSCW*, 314-323, 2002.

[3] Fass, A., Forlizzi, J., Pausch, R. MessyDesk and MessyBoard: Two designs inspired by the goal of improving human memory. *Proc ACM DIS*, 303-311, 2002.

[4] Fitzpatrick, G., Kaplan, S., Mansfield, T., Arnold, D., Segall, B. Supporting public availability and accessibility with Elvin: Experiences and reflections. *J CSCW* 11(3), 2002.

[5] Fujima, J., Lunzer, A., Hornboek, K., Tanaka, Y. Clip, connect, clone: Combining application elements to build custom interfaces for information access. *Proc. ACM UIST*, 175-184, 2004.

[6] Greenberg, S., Rounding, M. The Notification Collage: Posting information to public and personal displays. *Proc ACM CHI*, 515-521, 2001.

[7] Lessig, L. Free culture: How big media uses technology and the law to lock down culture and control creativity. The Penguin Press, 2004

[8] McEwan, G., and Greenberg, S. Supporting Social Worlds with the Community Bar. *Proc ACM Group*, 2005.

[9] Microsoft, Inc. Microsoft Alerts content providers: Sign up for Microsoft Alerts now! March 28, 2005. www.microsoft.com/net/services/alerts/.

[10] Olsen, D., Hudson, S., Verratti, T., Heiner, J., Phelps, M. Implementing interface attachments based on surface representations. *Proc ACM CHI*, 191-198, 1999.

[11] Richardson, T., Stafford-Fraser, Q., Wood, K. & Hopper, A. Virtual Network Computing, *IEEE Internet Computing*, Vol.2 (1), 33-38, 1998.

[12] RSS-DEV Working Group. RDF Site Summary (RSS) 1.0. http://web.resource.org/rss/1.0/spec. 2000

[13] Sugiura, A. and Koseki, Y. Internet scrapbook: automating Web browsing tasks by demonstration. *Proc ACM UIST*, 1998.

[14] Tan, D., Meyers, B., Czerwinski, M. WinCuts: Manipulating arbitrary window regions for more effective use of screen space. *Extended Abstracts ACM CHI 2004*.

[15] Zhao, Q., Stasko, J. What's happening? Promoting community awareness through opportunistic, peripheral interfaces. *Proc Advanced Visual Interfaces*, 2002.