Tracking Visual Differences for Generation and Playback of User-Customized Notifications

Saul Greenberg and Michael Boyle
Department of Computer Science, University of Calgary
Calgary, Alberta Canada T2N 1N4
E-mail: {saul, boylem}@cpsc.ucalgary.ca

ABSTRACT

Notification systems alert individuals or groups of changing information that is of interest to them. The problem is that it is difficult for people to gather notifications of personal interest; they must either rely on the generic offerings of the information provider, or construct their own services through coding. In this paper, we contribute a simple yet effective method that lets people create custom notification elements by *image assembly*, where notifications are triggered through *visual differencing*. First, after finding information of interest on a web page, the person constructs a visual collage selected from regions on the page. These are regions of the fully rendered bitmap view of the page i.e., they are not coupled to the page's underlying HTML markup. The composite image created from this collage will be used to assemble a notification of relevant changes to that web page. Second, the person specifies one or more regions on the page that will be compared for visual differences over time, and how often the page should be revisited to check for these differences. The system will automatically generate a notification (the composite image plus a title and timestamp) when differences go beyond a user-provided threshold. Finally, the person can view the notifications in several ways: as only the most recently changed version (to illustrate current state), or as an image history that can be either browsed individually or played back as a continuous video stream (to see changes over time).

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

General terms: Design, Human Factors

Keywords: Notifications, information customization.

INTRODUCTION

Notification systems let people selectively subscribe to personally interesting and dynamically changing information sources. The system then delivers relevant changes of that information – notifications – to those people. The actual information can be quite varied, with typical examples including stock prices, currency values, weather reports, traffic conditions, on-line status of people, new postings to newsgroups, sports reports, flight status, upcoming meeting appointments, webcam snapshots, and so on.

Many of these systems – especially the commercial ones – rely on a content provider to supply notifications. Often, however, these content providers offer only a few varieties of notifications. 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.

Even when a person finds matching content, problems remain. Content providers will charge for notifications – fees, advertisements, requirements for account creation (exploited for mailing lists), restrictions to a particular service (e.g., schedule alerts tied to the provider's calendaring system) – that may deter a person from subscribing to that service. In addition, the provider may not deliver notifications at a rate or in a form that satisfies the demands of that person (e.g., MSN currently delivers weather warning at prespecified times *vs.* when the weather changes).

In contrast, several systems – mostly research ones – allow people to create their own custom notification elements [2,3,5,7,14,18]. Yet creation is usually restricted to technically savvy programmers. They need to know how to tap into web services or other raw information sources; they need to parse this information; they need to program behaviors; they need to build specialized interface that displays the notification data in the end system. While custom coding is a powerful tool for prototyping and demonstrating different types of notifications, it is clearly not an option for the vast majority of people.

Consequently, our research goal was to develop a method that would let people (who are not programmers) generate their own custom notification elements. We wanted these people to be able to easily specify:

- an information source to be tracked, where that information appears in some form on a standard web page;
- the criteria for determining when changes in these sources are significant enough to warrant notification;
- how this information is be presented visually; and
- several methods for reviewing the notification history.

The remainder of this paper details how this goal was Ci

Cite as:
Greenberg, S. and Boyle, M. (2005) Tracking Visual Differences for Generation and Playback of User-Customized Notifications. Report 2005-777-08, Department of Computer Science, University of Calgary, Calgary, Alberta Canada. T2N 1N4. April.

achieved. To preview our approach, a user marks regions of interest on a web page, which are automatically stored as bitmaps. The system then periodically revisits that page at a user-specified time interval to see if the current regions visually differ from the stored ones by a user-determined threshold. If they do, the system generates and delivers a notification, which is itself automatically assembled as a collage of user-specified image regions from that page. The user views this notification in a form that best fits the type of information contained within it: a single snapshot, a browsable history of consecutive snapshots, or even a playable video stream.

RELATED WORK

Information sources for notifications. Several well-known methods are available for producing and/or publishing information that will eventually become a notification. First, various applications have built-in methods to generate notifications; a common example is a meeting calendar that publishes upcoming meeting times. Second, many commercial content providers now let people subscribe to 'alerting services', e.g., Microsoft's alerting service offers a myriad of alerts from various 3rd party companies [10]. Third, most research-based notification systems let programmers custom code hooks into their own information source. For example, users of the Notification Collage can code new multimedia notification elements as COM objects, which are registered and dynamically incorporated into the running system [7,14]. Similarly, users of MessyBoard can change its run-time behavior through Python scripting [3,4]. Finally, standards are being developed for alerts, where the information source can be accessed and interpreted by any system. For example, RSS is a set of XML elements for content syndication and distribution that are widely used by many news providers [15]. An RSS XML document describes channels of items and the URLs by which these items are retrieved. Yet, all these systems limit user customization of an information source in one way or another. The information source is fixed in some. Others require coding to add information sources or to interpret them and "translate" them into a form already recognized by the system. Invariably, most systems require proprietary software to generate and interpret the notifications.

Notification servers. These distributed computing systems provide the technical infrastructure for routing content from an information source to the users who want to be notified about it. In a typical scenario, an 'information producer' monitors the state of the source and sends updates about changes to this state to a centralized notification server. Consumers are notified whenever updates arrive, and can present that information in any form they wish. On the commercial side, some notification servers are inextricably linked to the software generating and displaying the notifications, while others are proprietary / paid services available only to specific content providers. However, many research systems work as independent, repurposable services. For example, with the Elvin Event Notification Service, producers publish notifications to Elvin as simple data types with named attributes, while consumers subscribe to sets of notifications using Boolean subscription expressions [5]. The Collabrary shared dictionary system allows arbitrary multimedia data to be published, where it is stored in a central repository [1]. Because of this storage, notifications in the Collabrary can be delivered to subscribers not only when they occur but also at a later time (e.g., as updates received when a subscriber 'logs on' to the service).

Notification displays. Many alternatives for how a notification is displayed to the user have been previously offered and explored. One approach leverages the capabilities of existing messaging applications. For example, a notification can be wrapped as an email message, which is then delivered and presented as conventional email message in the user's inbox. Similarly, it can be delivered as an SMS message to the user's mobile phone. Another display approach recognizes that each kind of notification has unique characteristics, and thus should be presented in its own interface. Examples include tickertapes and alerting icons [5], 'toasts' that appear and disappear on the display (e.g., as used in nearly all popular email and instant messenger applications), a series of windows that progressively reveal detail as they are queried in depth [2], and even as physical appliances [8]. A third approach aggregates the display of multiple notifications inside a container: e.g., a side bar [2]; a collage of visual elements [7,14,3,4,18]; or a tickertape that cycles through multiple notifications displaying each one by one [5].

Custom notifications. By customization, we mean that users can configure notifications to come from an arbitrary information source, where they also have considerable freedom to indicate how notifications are to be displayed. Some systems let users create arbitrary notifications as long as the information source belongs to a (usually very restricted) set of well-known data types. For example, both the Notification Collage [7,14] and Messyboard [3,4] let users publish arbitrary text, pictures and web links as notifications by manually dragging and dropping information onto the interface[1]. As previously mentioned, when the information source is not one of the well-known types, further customization requires scripting or programming [2,4,7,14,]. This requirement makes customization generally out of the reach of most users. A further caveat is that some systems cannot incorporate these custom notifications without recompilation, and that many systems (especially commercial ones) disallow arbitrary alerting services to be added by the general public.

Custom displays. Most notification systems either do not permit customization of the notification display or require scripting / programming to achieve this kind of customization. However, there are a few related systems - not notification systems - that allow users to customize an information display. With Tan et. al.'s WinCuts [17], 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

[1] In these two systems, notifications happen only when a user explicitly adds the information to the interface through manual action; hence, these systems are more akin to multimedia groupware (communication-oriented) than a true notification service (alerting-oriented).

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 [6]. Hunter Gatherer lets users create content collections by harvesting page snippets as they navigate the web [16] (this is done by HTML extraction *vs.* images). More generally, Olsen et al. [12,13] argue that surface representations of applications – the display images that the user sees – can be leveraged as a uniform service that can be applied to many application settings. Later in this paper, it will be apparent why these ideas of somehow clipping information from source documents are highly relevant to our own method that allows users to easily create custom notifications.

In subsequent sections, we detail how we have combined and extended aspects of this related work to create a notification system that has users specify regions of interest taken from the rendered display of a web page, where the system then periodically polls the page to perform visual comparisons over time to determine when notification generation is warranted.

WHAT THE USER SEES AND DOES

Before delving into technical details, we describe the end user experience as she creates and displays custom notifications. For simplicity, we will walk through a detailed example of how 'Jane' creates a notification that displays new postings to the Slashdot web site. Later examples will summarize how the same steps can be applied to other styles of notifications.

We stress that describing this interactive process in an academic paper – with text and static diagrams – makes the system appear more complex to use than it actually is; in practice, the sequence of user actions outlined below are easily accomplished in under a minute. As well, portions of these steps are optional; the defaults for most settings usually suffice for most situations.

A. Creating the Custom Notification

Motivation. Jane is interested in the postings on Slashdot, a very popular 'news for nerds' web site. However, she finds it tedious to regularly revisit the Slashdot web page: there may be no new postings, the many postings are not particularly relevant, or through neglect she has missed earlier postings and has to scan for them. Consequently, she decides to create a custom notification to Slashdot by using the *Notification Editor*. Figure 1 annotates this editor, and displays the final results of her editing actions.

Step 1. Selecting the information source. Jane types in the Slashdot URL in the *Address Bar*. The current Slashdot web page appears in editor's left pane as a bitmap image.

Step 2. Specifying change region(s) of interest. Jane knows that changes to a web page can occur for many reasons: advertiser content may differ between visits; animations and scripts may be triggered at different times; new but irrelevant information may be posted. What she is mostly interested in is the lead article at the top of the page; she only wants to be notified if that article has changed.

Using the *Tool palette* at the top of the editor, she selects the *Change region* tool that lets her specify what regions of interest should by later monitored by the system for changes, as determined by visual differencing between page revisits. This tool lets her drag out a rectangle in the *Content editor* (left side, Figure 1) that surrounds the content to track. In this case, she positions a change region so that it surrounds the caption of the first article, as she expects this to change when new postings appear.

She then adjusts the properties of this region using the controls in the *Region properties* pane (top right, Figure 1). First, she labels this change region 'Article title' for easy reference – this label then appears as part of the change region on the Content editor. Next, using the *Change trackbar* she sets a change threshold[2], 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 generation of a notification. In this case, she leaves it at its smallest setting, as she believes that any visual changes in this region is likely the result of a new posting.

Jane can continue this process by creating other change regions of interest on this page; in this example, the single change region suffices. If multiple change regions are specified, each may be assigned its own change threshold: in the current implementation, a notification is triggered for the whole page if any one of the change regions triggers a notification.

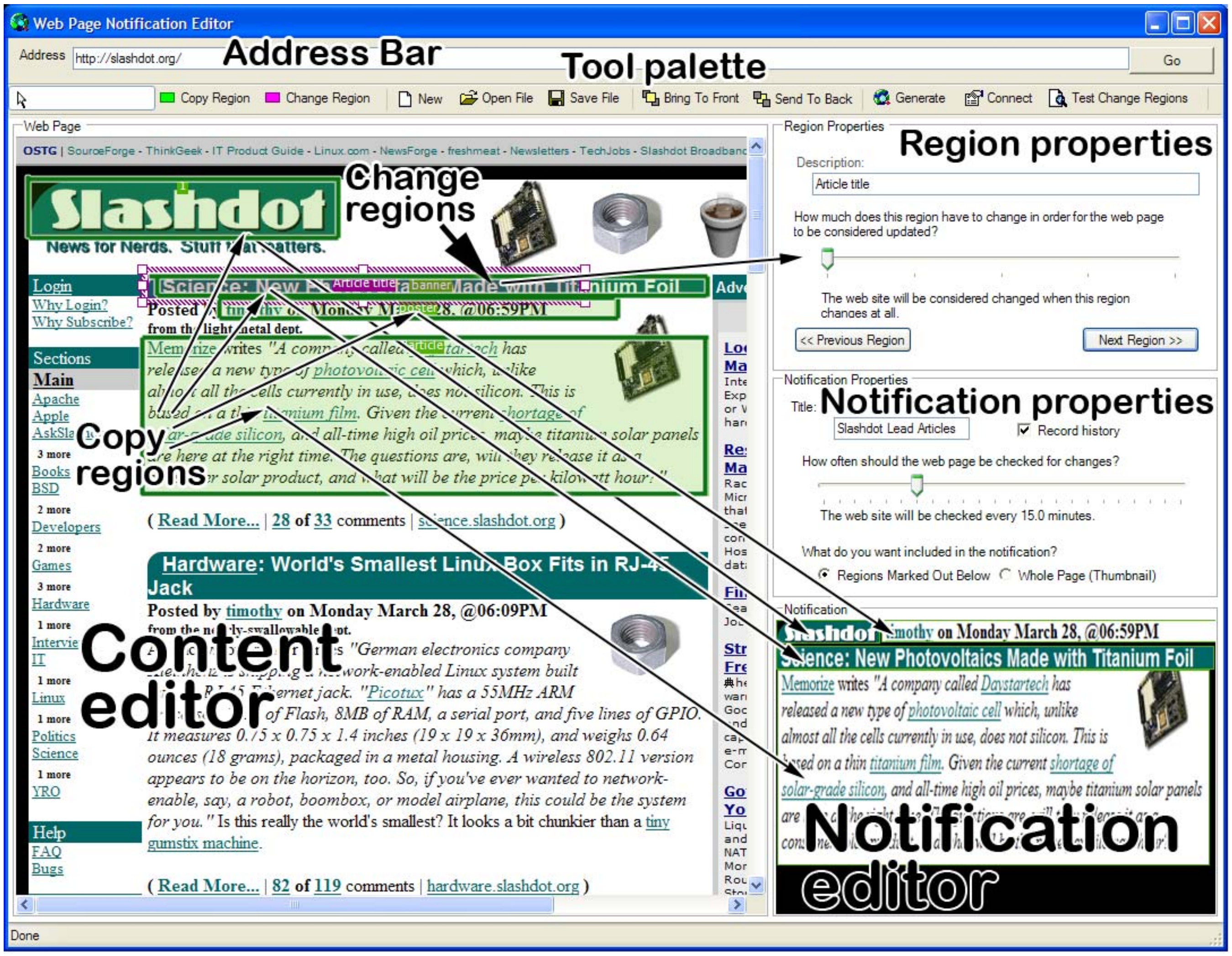
Step 4. Specifying the notification title and history. Jane is now ready to configure the notification. Using the controls in the *Notification properties* pane (mid-right, Figure 1), Jane specifies a title of the notification "Slashdot Lead Articles". This title will eventually be displayed as part of the notification that appears in a Notification viewer.

Jane also leaves the *Record history* checkbox checked. This means that the underlying system will save a history of all generated notifications, which she will be able to later review via the Notification viewer.

Step 5. Specifying revisit frequency. While in the notification properties pane, she adjusts the contained trackbar to specify how often the web page should be checked to see if the change regions have changed, e.g., every few minutes. When the system is instructed to track this page, it will automatically revisit the page based on this interval and will compare the bitmaps defined by the changed 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. Based on her knowledge of Slashdot postings as well as her own very high interest in Slashdot, she sets the revisit frequency for 15 minutes.

Step 6. Assembling a notification. When the system generates a notification (because it detected changes) it creates and publishes a scaled down version of the whole web page as a high quality thumbnail. However, Jane would rather see a readable fragment of the most recent posting in the

[2] To fine tune the threshold, the *Test change regions* command on the Tool palette displays before, after, and 'differenced' images of the regions between successive page visits in a transient window (not shown).



Address: http://slashdot.org/

Description: Article title

How much does this region have to change in order for the web page to be considered updated?

The web site will be considered changed when this region changes at all.

Title: Slashdot Lead Articles ☑ Record history

How often should the web page be checked for changes?

The web site will be checked every 15.0 minutes.

What do you want included in the notification? ● Regions Marked Out Below ○ Whole Page (Thumbnail)

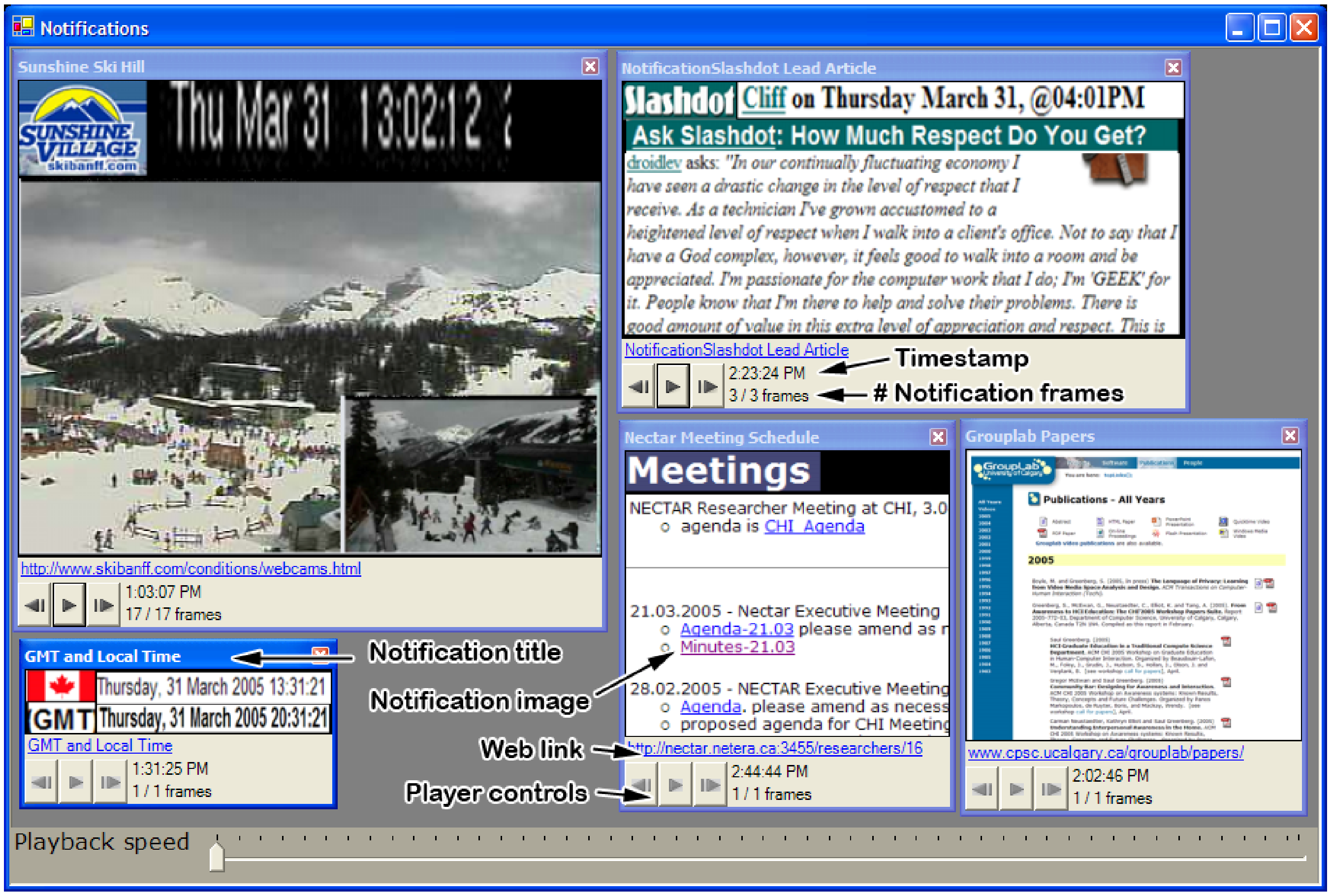
Figure 1: The Notification Editor.

notification. This will allow her to read its partial contents: if it is interesting and she would like further details she could then opt to visit the actual web page. Consequently, she decides to construct the notification visuals by assembling a collage of selected regions from the Slashdot page.

Jane selects the *Copy region* tool from the Tool palette. As with the Change region tool, she can drag out rectangles to specify regions on the web page displayed in the content editor. As she does so, these regions are automatically copied to the *Notification editor* pane (lower right, Figure 1). These copied images are still linked to the original regions in the content editor: any changes made to the rectangle that defines these regions (size, location) will be immediately reflected in these copies. At any time, she can rearrange these image fragments in the Notification editor pane. She can select, move, and resize these copied images (resizing scales the image to a larger or smaller size). She can also overlap images, where she can fine tune which image is on top by using the *Bring to front / Send to back* tools in the Tool palette. This visual layout of copy regions in the pane becomes the notification presented to her when a change to the page is detected. Figure 2 shows an example.

Figure 1 shows the notification that Jane constructed in the Notification editor pane. It has four copy regions: the Slashdot logo at the upper right, the title of the lead article, the poster's name and time, and about 7 lines of the body of the lead article. Full sized versions of these are automatically copied to the thumbnail palette. (The arrows in Figure 1 show the source copy regions in the Content editor and where they appear in the Notification editor.) As illustrated in the Notification editor in the figure, she creates a visual banner by abutting the regions containing the Slashdot logo and the poster's name/time, and by resizing the Slashdot logo so that it is the same height as the poster's line. She then moves and scales the region containing the lead article title to fit below this banner. Below that, she repositions the region containing the article body, and shrinks it down to a size that fits nicely will still being readable.

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. By selecting the *Save file* tool on the Tool palette, she can save this specification (stored as an XML file). She can later reload this or another file for



Slashdot Cliff on Thursday March 31, @04:01PM

Ask Slashdot: How Much Respect Do You Get?

droidlev asks: "In our continually fluctuating economy I have seen a drastic change in the level of respect that I receive. As a technician I've grown accustomed to a heightened level of respect when I walk into a client's office. Not to say that I have a God complex, however, it feels good to walk into a room and be appreciated. I'm passionate for the computer work that I do; I'm 'GEEK' for it. People know that I'm there to help and solve their problems. There is good amount of value in this extra level of appreciation and respect. This is

NECTAR Researcher Meeting at CHI, 3.0
- agenda is CHI Agenda

21.03.2005 - Nectar Executive Meeting
- Agenda-21.03 please amend as r
- Minutes-21.03

28.02.2005 - NECTAR Executive Meeting
- Agenda. please amend as necess
- proposed agenda for CHI Meeting

Figure 2: The Notification Viewer

further editing by selecting the *Open file* tool. She could also send this file to a friend (e.g., by email) so her friend can get similar notifications of changes to Slashdot. Alternatively, her friend could further edit and customize this description to meet his own needs.

B. Publishing / Posting Notifications

Jane needs to activate the notification generator. This can be done through an external program or within the editor. In this case Jane will do it from the editor. First, she selects the *Connect* tool on the Tool palette, and through a simple dialog box she specifies the URL location of the notification server, e.g., tcp://jane.site.ca:server. Next, she starts the notification generator by selecting the *Generate* tool.

At this point, the Notification generator will use the notification description to determine if a relevant change has occurred on the web page. If it has, the Notification generator will assemble a notification image, and publish it to the notification server which in turn will publish it to Notification viewer clients. For immediate local feedback, the Content editor pane and the Notification editor pane are updated to display the most recent revisitation of the page and the corresponding generated notification.

Of course, other notifications may be published to the server at the same time. The server stores this information centrally, and will publish it to the Notification viewer clients (described next) that are connected to it.

C. Viewing Notifications

Jane can now view this (and other) notifications whenever she wishes, and from whatever machine she happens to be working on.

Jane starts the Notification viewer client (Figure 2), which connects to the notification server. She immediately sees and scans the four visible notification items, including the Slashdot notification she had specified: these all represent items that have visually changed over the course of time. All notifications also contain a link to the original web page (the *Web link* URL under the image). Clicking this link opens the page in her web browser.

Jane can manipulate these notification items in several ways. Each is shown in its own small window, which she can reposition and resize: the contained image is scaled to fit. She can close the notification window when she has read it: it reappears when a change is next detected and a new notification generated.

Jane also notices that the video player controls (bottom of the notification item) within the Slashdot notification are active, and that 3 notification frames are available for viewing. These frames represent notifications received earlier and are maintained in a history. Jane wants to see if any of these earlier Slashdot postings are interesting, so she uses the video player controls to step backwards through them, one at a time. This allows her to rapidly scan the past postings to see if she has missed anything of interest.

Jane had previously created a notification to her local ski hill (top left). This notification displays two web camera images as a picture in picture, captured every few minutes as part of the change monitoring process. Jane hits the video *Play* button, and sets up fast speed video playback by adjusting the replay speed through the *Playback speed* trackbar at the bottom of Figure 2. She notices that the ski line-ups are consistently long and that the weather appears to be worsening. While she was thinking of taking the afternoon off to go skiing, she decides the ski hill is just too busy and that white-out conditions are likely to develop by the time she gets to the hill.

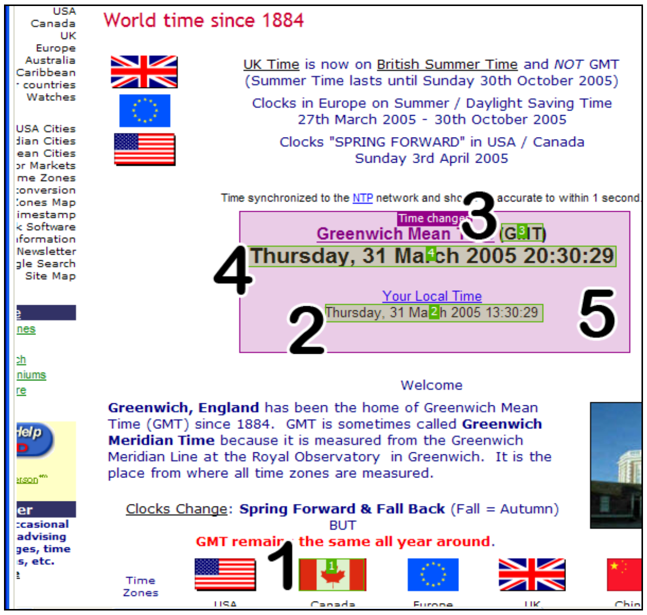
Over the course of the day, these notifications are updated and new ones appear as information is published to the notification server. In this way, Jane quickly sees new postings to the Slashdot news site as well as changes to the other pages. If she steps away from her machine, she can catch up on any missed notifications when she returns by replaying the individual notification histories.

THE RANGE OF NOTIFICATIONS

To illustrate the richness of this method, we detail how the other notifications in Figure 2 were constructed. While the construction, posting and viewing mechanism is identical to the steps and actions taken above, the uses for these notifications are quite different.

Constructing a clock. The www.greenwichmeantime.com web page displays both the Greenwich Mean Time (GMT) and the local time – a cropped fragment of this page is visible in Figure 3, with regions annotated by numbers. The simple minute-sensitive clock notification at the bottom of Figure 2 was constructed from this source. The change region '5' over the time area specifies what to monitor for changes. The page revisit frequency was set to once per minute. The notification was assembled from scaled copy regions 4 and 2, which hold the GMT and local time visuals, as well as identifying labels harvested from regions 3 and 1. Because there is no value in saving a history of these notifications, the *Record history* option was turned off in the Notification properties editor. Note that the time notification in Figure 2 includes only essential information relevant to its creator, and it is quite small and compact when compared to the original web page in Figure 3.

Constructing a publication and meeting notifier. Many researchers, including ourselves, infrequently update a publication list as they produce papers. Yet seeing if new papers are available by revisiting these pages is tedious. However, a custom notification can be easily created to monitor these pages. The notification titled 'Grouplab papers' shown at the bottom right of Figure 2 is one example. To create this notification, a change region was dragged around the current year's publication list. The revisit frequency was set at once a day, and the descriptive title entered. Instead of creating custom notification visuals by selecting copy regions, it is acceptable in this case to use a thumbnail of the whole page as the notification image. This is selected using the radio buttons at the bottom of the Notification properties pane in Figure 1. When the notification



World time since 1884

UK Time is now on British Summer Time and *NOT* GMT
(Summer Time lasts until Sunday 30th October 2005)

Clocks in Europe on Summer / Daylight Saving Time
27th March 2005 - 30th October 2005

Clocks "SPRING FORWARD" in USA / Canada
Sunday 3rd April 2005

Time synchronized to the NTP network and sho accurate to within 1 second.

Greenwich Mean (G IT)
Thursday, 31 Ma ch 2005 20:30:29

Your Local Time
Thursday, 31 Ma h 2005 13:30:29

Welcome

Greenwich, England has been the home of Greenwich Mean Time (GMT) since 1884. GMT is sometimes called **Greenwich Meridian Time** because it is measured from the Greenwich Meridian Line at the Royal Observatory in Greenwich. It is the place from where all time zones are measured.

Clocks Change: **Spring Forward & Fall Back** (Fall = Autumn)
BUT
GMT remair the same all year around.

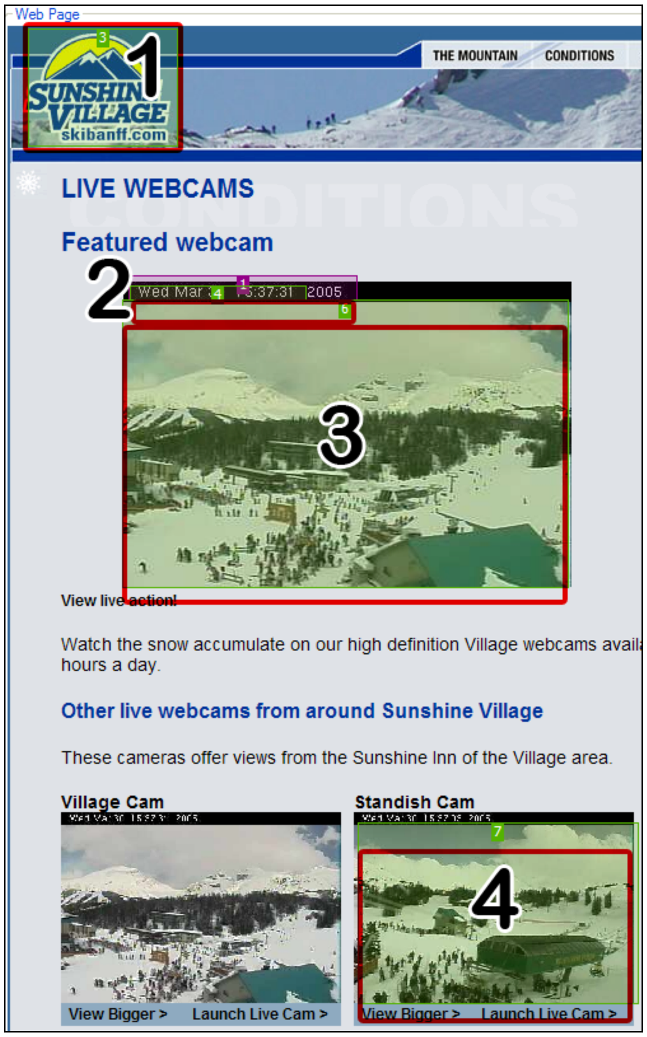
Figure 3: Source document, including numbered copy/change regions, used to build the Greenwich Mean Time notification in Figure 2.

generator detects the (rare) change, it automatically produces the thumbnail image seen in Figure 2's notification. Along with the title, this provides enough information to indicate what this notification is about. The person can then decide to visit that site.

The meeting notification seen in the middle-bottom of Figure 2 was constructed in a similar fashion, except in this case the user opted to assemble a readable notification that includes the most recent held and scheduled meetings.

Constructing a ski webcam video. The Sunshine village ski area features a web page (Figure 4) that includes several webcam images showing activity at the ski hill over the day. These cameras do not run continuously, e.g., they are sometimes turned off at night or for unpredictable reasons. In such circumstances, the web camera image is not updated. It is easy to create a custom notification - similar to the other notifications - that contains just the webcam image, accompanied by some extra regions to identify the web site (e.g., the ski hill logo). However, the threshold for changes between successive webcam images can be set somewhat higher than that used in the other notifications, so to avoid sending out notifications if there are no interesting changes in the scene. For example, two successive images will almost certainly have minor differences between them due to noise in the camera, and these do not warrant a notification.

Figure 4 shows the source web page as well as the four copy regions used to create the notification visible at the top left of Figure 2. What is especially interesting is how the user created a picture-in-picture effect in the notification by overlaying a scaled down version of one of the webcam images (region 4) atop another webcam image (region 2).



LIVE WEBCAMS

Featured webcam

View live action!

Watch the snow accumulate on our high definition Village webcams avail hours a day.

Other live webcams from around Sunshine Village

These cameras offer views from the Sunshine Inn of the Village area.

Village Cam

Standish Cam

Figure 4: Source document used to build the Sunshine Live Webcam notification in Figure 2.

On the viewer side, the viewer will not only see these changes, but can play back a video of all activity captured by the webcams over the course of the day. If the playback speed is set to its maximum, the day's activities (including changes in weather) will be compressed to a minute or so of video.

IMPLEMENTATION.

The architecture for this system is reasonably straightforward, but is described in detail to expose its strengths and weaknesses and to encourage replication.

The Notification Editor, as seen in Figure 1 and represented at the top left of Figure 5, 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, is described in Table 1. As can be seen, the specification includes the *Url* of the page to be monitored, the *Monitoring frequency* describing how often that page should be checked for changes, and a list of *Monitored regions* that specify the rectangular regions on the page to check for bitmap differences and a *Threshold* indicating how much a region must change for a notification to be triggered. It also contains 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.

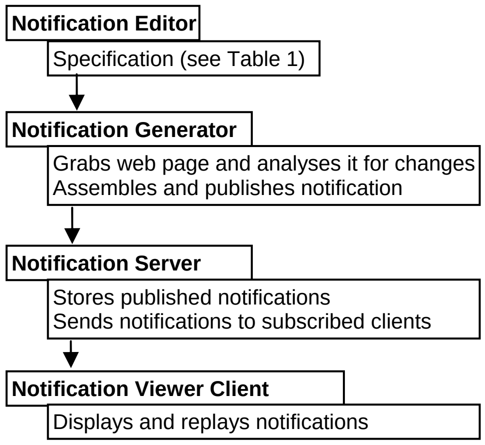

Notification Editor
Specification (see Table 1)
Notification Generator
Grabs web page and analyses it for changes
Assembles and publishes notification
Notification Server
Stores published notifications
Sends notifications to subscribed clients
Notification Viewer Client
Displays and replays notifications

Figure 5: Basic architectural components

```
<WebPageNotification>
  <Url>URL of source page</Url>
  <MonitoringFrequency>
    time interval to check page
  </MonitoringFrequency>
  // a list of Monitored Regions on the web page; only one is shown here
  <MonitoredRegions>
    <MonitoredRegion>
      <UpperLeft>X,Y  corner of monitor region on web page</UpperLeft >
      <Size>Width, height of monitor region </Size>
      <ChangeThreshold>
        Change threshold between bitmaps needed to trigger notification
      </ChangeThreshold >
    </MonitoredRegion>
  </MonitoredRegions>

  <Title>Title of the notification</Title>
  <HistoryLength>
    The length of the notification history to be maintained
  </HistoryLength >
    // a list of CopyRegions; only one is shown below, Each source region
    // on the web page has a matching destination region on the notification
  <CopyRegions>
    <CopyRegion>
      <SourceOnWebPage>
        <UpperLeft>X,Y  corner of copy region on web page</UpperLeft >
        <Size>Width, height of coopy region </Size>
      </SourceOnWebPage >
      <DestinationOnNotification>
        <UpperLeft>X,Y  corner of image in notification area</UpperLeft >
        <Size>Width, height of image in notification area</Size>
      </ DestinationOnNotification>
    </CopyRegion>
  </CopyRegions>
  <PngImageData>
    Compressed bitmap of the source webpage
  </PngImageData>
<WebPageNotification>
```

Table 1: XML of specification produced by the editor

Aside from its user interface, the only special feature required by the Notification editor is the ability to render a web page into a bitmap and perform various image capture and manipulation functions. To simplify this task, we used the Collabrary Toolkit [1], which provides an easy-to-use API for capturing, processing and distributing multimedia data. The Collabrary includes the ability to do efficient off-screen rendering of web pages, basic image manipulation such as cropping, scaling, and the visual differencing used to compare image regions.

The Notification Generator can be invoked from the notification editor, or can run as a stand-alone application. As seen in Figure 5, it uses the specification produced by the Notification editor. The Notification generator renders the web page off-screen into a bitmap at the specified monitoring frequency. After grabbing the web page, it compares the monitored regions of the current page with the last one it has seen (if it is the first image grab, it will compare it to the original web page image stored in the specification's *PNG-compressed image data* field).

If a change is detected in a monitor region 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 capturing, locating and scaling the copy regions on the source page into the destination notification.

Finally, the generator publishes this information to a notification server (Figure 5). The published data is very 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 history length, then the oldest notification is removed before a new one is inserted.

The Notification Server. We use our own GroupLab .Networking notification server system, which is a redesigned version of the notification capabilities found in the Collabrary [1]. It has many features well suited to the task of marshalling, storing and efficiently distributing multimedia information to subscribed clients. In particular, it: allows notification contents to be organized in a hierarchy of key/value pairs; automatically marshals both primitive data types and complex types like compressed images; and, can maintain lists of notification values as easily as storing a single value. While other notification server systems could have been used instead, they may lack the GroupLab.Networking features that make it very easy to implement the Notification generator. However, because of the simplicity of the data produced by the Notification generator, the required capabilities could be simulated atop another notification server system, or the generator rearchitected as a workaround.

The Notification Viewer Client. The final component in our architecture is the presentation client. As seen in Figure 5, the client consumes and displays notifications generated by the server. To do this, the client connects to the notification server, subscribes to the key/value pairs that hold the relevant notification data, and displays the information therein in the interface as each new notification arrives. 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 in our notification client 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.

Similar to the Notification Collage and MessyBoard, our viewer client interface presents individual notifications as a collage of possibly overlapping items positioned in a large window (Figure 2). However, there is nothing in our method that insists on this arrangement, and other quite different interfaces could have been constructed instead.

DISCUSSION

Power. The power of our image-based visual differencing method is its simplicity. First, it leverages the *surface representation* strategy of Olsen et al., [12,13] 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 markup), surface representations are designed to be understood by the user. Consequently, we base our notification creation strategy on image region selection, copying and differencing, as these are notions that users will easily grasp. For change regions, users intuitively know 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.

Second, our strategy works with a very broad set of arbitrary data sources because it leverages bitmap-based renderings of the information source as a lowest common denominator for information representation and extraction. Regardless of how a web page is produced or what HTML standard the browser follows, the page is ultimately rendered on-screen as an image. Our approach needs no *a priori* knowledge of how the image is rendered: as long as there is some mechanism to capture the images in the first place (i.e., render the information source as a bitmap) the system will work, taking care of manipulation, extraction, and comparing visual differences. As a lowest common denominator, our image-based approach is not limited to web pages and could easily be used with screen-grabs of running GUI application windows as in the WinCuts system [17]. Of course changes to the Notification editor will be needed: e.g., the editor will need to grab screenshots of GUI application windows instead of web pages, and the user interface must provide a mechanism for the user to select a window on the screen to capture.

Third, because the data generated is comprised of only a few text fields and a list of images, it is easy to collect and distribute. We expect that our system could be implemented atop any notification service, including an SQL database. Of course, the particulars of how connections are made to the service and how the text and image data generated are translated into the data standard required by it (e.g., as an XML document *vs.* a variable length blob of binary data) will require (straightforward) changes to the Notification generator and viewer.

Fourth, since our strategy generates and presents the same kinds of data for any arbitrary information source, the presentation of a notification becomes decoupled from the type of information source. The Notification viewer only needs to know how to display some simple textual attributes – title, time, url – and an image. Because bitmaps are used as the representation for all information sources and all notifications, this strategy blindly incorporates any notification the user may want to create.

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, etc. Resulting notifications are very compact.

Finally, the user experience – while difficult to describe in an academic paper – 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 are generated. The user can see changes as they unfold over the course of a day. The ability to play back the history of a web site as a video stream adds information not normally discernable in a web page's static images.

For example, we ran the Sunshine Ski Hill notification seen in Figure 2 over 24 hours, capturing changes every two minutes. When the history of notifications generated over this period was played back at an accelerated rate, some spectacular glimpses into life at the ski hill were revealed: a buzz of activity during the night as the workers groomed the slopes; dawn breaking over the Canadian Rockies, with clouds rapidly moving across a golden sky; the sky darkening as a storm came in; crowds trickling in when the hill first opens in the morning, gradually surging into a rush at 10:00am; people congregating by the main lodge around noon and leaving near the end of the day. As dusk arrived, the cycle repeated. The source web page, created to provide a snapshot of hill weather and activity at an instant in time, is transformed by our system into a documentary of the patterns of life that emerge over the course of an entire day.

Limitations. While powerful, there are several limitations to our method. The image-based approach to creating a notification relies on a web page retaining its formatting and layout between visits. Fortunately, this is often true. The top of a page normally comprises standard information repeated across all visits, e.g., page titles, logos, banner advertisements: these are typically of fixed heights. Content and navigation side bars are also typically fixed in width. Designers know that only 10% of users scroll beyond the information visible on the screen when the page comes up [11]; thus they will typically place critical changing information – the kind that a user would likely find valuable as a notification – in a relatively fixed location immediately beneath page headers so that it appears 'above the fold'[3].

However, users will not be able to create a notification that captures a region of changing information whose position floats on the page. For example, consider a page with two variable length segments of text, one above the other, that may change between visits. If the user is only interested in the second text segment, he has no easy way of determining where that text will appear. In this case the only strategy would be to capture the whole page as a thumbnail, or to mark out a region large enough so that it anticipates where that text may appear.

Our system works with rendered bitmaps of web pages and, as a result, hyperlink visuals on the page are not navigable. While we could have used a WinCuts / Clip, Connect, Clone approach to link user input actions on the image to the source web page [6,17], we felt this unnecessary; it introduces the potential for errors as the page contents can change when the user clicks a hyperlink. Instead, we rely on an external web browser to support navigation, which can be easily raised by clicking the web link in the Notification viewer (Figure 2).

Users can scale copy regions in the Notification editor however they like, but they could produce images that are scaled too small to read legibly. Making the display window in the Notification viewer larger will not help if the original notification has insufficient resolution. To mitigate this, the Content editor always copies the full sized image into the Notification editor (Figure 1), which encourages the user to create legible notifications. While the notification viewer may later scale the generated notification to fit a small window, it can be expanded to its full resolution when (say) the user resizes the window to be larger. An alternatively approach may be to use a sidebar metaphor [2] in which the bar displays the assembled collage reduced to fit the display space allocated for it but then support 'drill-down' by showing each individual copy region at its original large size and aspect ratio in a 'tooltip grande'.

One must also take care with the history list size. In our implementation, it is possible to create a notification that takes snapshots of a frequently updated web page (such as the Sunshine ski hill page) every 15 seconds. The storage requirements for maintaining a history of such notifications could rapidly swell to unmanageable proportions. Consequently, in our system, we limited the maximum size of a single history list to 200 notifications, where old notifications are discarded as new ones arrive. Given the a size of our images and that they are compressed, we anticipate minimal resource usage.

[3] Our system performs off-screen rendering of a web page into a container of a specified (fixed) width. Thus the reformatting that normally results from changing a browser window size is not a problem.

Next, web pages can be idiosyncratic. Many embed programs (e.g., Java applets and Flash animations) or use client-side scripting to finalize document construction after the page is loaded. On these types of pages, grabbing an image after a 'document complete' event appears can result in a web page with missing elements. Adding a time delay of several seconds to let these elements load is a prudent workaround, but may still be unreliable. Other practical problems will make rendering bitmaps of web pages troublesome: password protected web sites, server redirection, popup browser windows, CAPTCHAs, client-side script errors, server-side web application errors, network connectivity issues, and so forth. In spite of these issues, a large number of web pages are suitable for our method.

Finally, there are the legal and ethical issues involved when information is clipped from a web site without permission. Some content providers believe that its information should only be shown in the context of the original web page; after all, many make their money through associated advertisements. The actual law varies from country to country and is currently hotly contested. While many consumer advocates believe that this use of material falls under 'fair use', there is a large lobby effort by corporations to make remixing of web material illegal [9]. Other ethical concerns arise from the capture of static information (such as webcam snapshots) as a replayable video, for the information revealed by the video may not be what the author intended. An example could be continuous webcam capture of a person's home (which some people post to the web) every 15 seconds.

CONCLUSION

This research makes several significant contributions. At the high level, we advance a novel method that uses visual differences in user-selected regions of a web page's surface representation – its bitmap image – to serve as the basis for a notification system: custom notifications are themselves harvested and assembled from user-specified regions on that page. While aspects of this resemble the clipping approach for custom interface design advocated by Tan et. al. [17] and Fujima et. al. [6], its application to notification generation gives a very novel and completely different effect. At a finer granularity, we also contribute:

- how custom presentations of notifications can be assembled from changes to these images (leveraging ideas found in [6,7]);
- 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.

Trying out this system is more fun than reading about it! Download and install it from http://www.url.to.be.supplied.

REFERENCES

1. Boyle, M., Greenberg, S. GroupLab Collabrary: A toolkit for multimedia groupware. J. Patterson (Ed.) Workshop Network Services for Groupware ACM CSCW'02.
2. Cadiz, J., Venolia, G., Jancke, G., Gupta, A. Designing and deploying an information awareness interface. *Proc ACM CSCW* 2002, pp. 314-323.
3. Fass, A., Forlizzi, J., Pausch, R. (2002). MessyDesk and MessyBoard: Two designs inspired by the goal of improving human memory. ACM DIS 2002, 303-311.
4. Fass, A., Pausch, R. (2002). Adding scripting to a public bulletin board. Poster at ACM UIST 2002.
5. Fitzpatrick, G., Kaplan, S., Mansfield, T., Arnold, D., Segall, B. Supporting public availability and accessibility with Elvin: Experiences and reflections. *J Computer Supported Cooperative Work* 11(3), 2002.
6. Fujima, J., Lunzer, A., Hornboek, K., Tanaka, Y. Clip, connect, clone: Combining application elements to build custom interfaces for information access. *Proc. ACM UIST* 2004, pp. 175-184.
7. Greenberg, S., Rounding, M. The Notification Collage: Posting information to public and personal displays. *Proc ACM CHI* 2001, pp. 515-521.
8. Greenberg, S. and Fitchett, C. (2001) Phidgets: Easy development of physical interfaces through physical widgets. *Proc ACM UIST* 2001, p209-218.
9. Lessig, L. Free culture: How big media uses technology and the law to lock down culture and control creativity. The Penguin Press, 25 Mar 2004 |
10. Microsoft, Inc. Microsoft Alerts content providers: Sign up for Microsoft Alerts now! March 28, 2005. www.microsoft.com/net/services/alerts/.
11. Nielsen, J. Original top ten mistakes in web design. Jakob Nielsen's Alertbox 1996. //www.useit.com/alertbox/.
12. Olsen, D., Hudson, S., Verratti, T., Heiner, J., Phelps, M. Implementing interface attachments based on surface representations. *Proc ACM CHI* 1999, pp. 191-198.
13. Olsen, D., Hudson, S., Phelps, M., Heiner, J., Verratti, T. Ubiquitous collaboration via surface representations. *Proc ACM CSCW* 1998, pp. 129-138.
14. Rounding, M. (2004) Informal Awareness and Casual Interaction with the Notification Collage. MSc Thesis, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada, April.
15. RSS-DEV Working Group. RDF Site Summary (RSS) 1.0. http://web.resource.org/rss/1.0/spec. 2000
16. schraefel, m.c., Zhu, Y., Modjeska, D., Wigdor, D., Zhao, S. Hunter Gatherer: Interaction support for the creation and management of within-web-page collecctions. *Proc ACM WWW* 2002, pp. 172-181.
17. Tan, D., Meyers, B., Czerwinski, M. WinCuts: Manipulating arbitrary window regions for more effective use of screen space. *Extended Abstracts ACM CHI 2004*.
18. Zhao, Q., Stasko, J. What's happening? Promoting community awareness through opportunistic, peripheral interfaces. *Proc Advanced Visual Interfaces,* 2002.