The Buzz: Supporting User Tailorability in Awareness Applications

James R. Eagan
eaganj@cc.gatech.edu

John T. Stasko
stasko@cc.gatech.edu

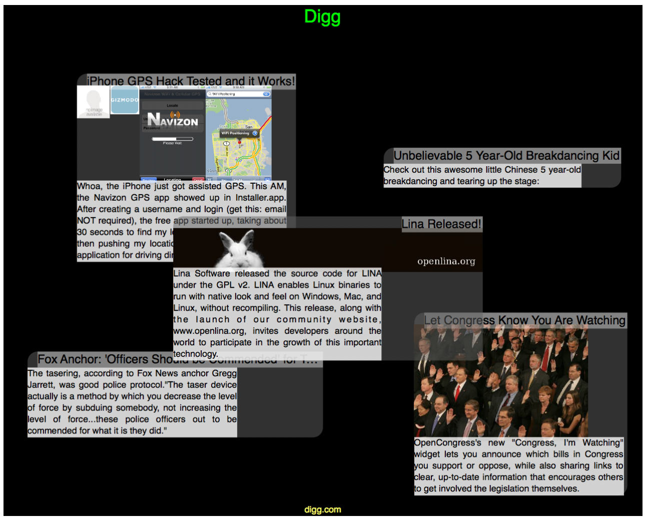
School of Interactive Computing & GVU Center
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA

ABSTRACT
Information awareness applications offer the exciting potential to help people to better manage the data they encounter on a routine basis, but customizing these applications is a difficult task. Most applications allow users to perform basic customizations or programmers to create advanced ones. We present an intermediate customization space and Cocoa Buzz, an application that demonstrates one way to bridge these two extremes. Cocoa Buzz runs on an extra display on the user's desktop or on a large shared display and cycles through different information sources customized by the user. We further demonstrate some of the customizations that have been made using this approach. We show some preliminary evidence to suggest that this approach may be useful at providing users with the ability to perform customizations across this spectrum.

ACM Classification Keywords
H.5.m Information interfaces and presentation (*e.g.*, HCI): Misc.

INTRODUCTION
The combination of abundant and ubiquitous access to data with falling costs of display technologies is forming an exciting opportunity to create new and interesting information awareness applications. These applications can gather data from a variety of sources, compound them together in interesting ways, and create new interfaces on the data. For example, mashups focus on taking data from one or multiple sources and integrating them together with a different interface, as in combining housing data from Craig's List with a mapping interface from Google [21]. Other applications focus on presenting the data via calm, ambient or peripheral interfaces [20] that may potentially be able to help people to better manage interruption [7].

CHI 2008, April 5 - 10, 2008, Florence, Italy.
Copyright 2008 ACM 1-59593-178-3/07/0004...$5.00.



iPhone GPS Hack Tested and it Works!

Whoa, the iPhone just got assisted GPS. This AM, the Navizon GPS app showed up in Installer.app. After creating a username and login (get this: email NOT required), the free app started up, taking about 30 seconds to find my l… then pushing my locati… application for driving di…

Unbelievable 5 Year-Old Breakdancing Kid

Check out this awesome little Chinese 5 year-old breakdancing and tearing up the stage:

Lina Released!

Lina Software released the source code for LINA under the GPL v2. LINA enables Linux binaries to run with native look and feel on Windows, Mac, and Linux, without recompiling. This release, along with the launch of our community website, www.openlina.org, invites developers around the world to participate in the growth of this important technology.

Let Congress Know You Are Watching

OpenCongress's new "Congress, I'm Watching" widget lets you announce which bills in Congress you support or oppose, while also sharing links to clear, up-to-date information that encourages others to get involved the legislation themselves.

Fox Anchor: 'Officers Should be Commended' for T…

The tasering, according to Fox News anchor Gregg Jarrett, was good police protocol."The taser device actually is a method by which you decrease the level of force by subduing somebody, not increasing the level of force...these police officers out to be commended for what it is they did."

Figure 1. Articles from the collaborative news site, Digg.com, as shown on Cocoa Buzz.

These personal information awareness systems, however, are inherently personal in nature. Such systems need to support some degree of customization in order to allow the user to tailor the content to his or her own personal interests. Supporting such customization, however, requires application designers to forge a delicate balance between the expressiveness of the customization system and the ease of using such a system. As a result, many systems focus on providing rudimentary customizations or advanced customizations, but do not focus on bridging the area in between. Thus, users can customize basic properties of awareness content, such as providing a postal (zip) code, an email address, or a list of stock symbols [8, 3, 5]. Or they can define complex visual or software programs to manipulate their data or presentation [24, 4, 8]. But performing intermediate customizations are either impossible or involve using a full-fledged programming environment.

We have created a customizable information awareness application, called Cocoa Buzz[1], that allows users to richly tailor their information awareness content, or channels. To

[1]Cocoa Buzz is a play on the caffeine jolt one gets from the drink and the UI toolkit on which the software is built.

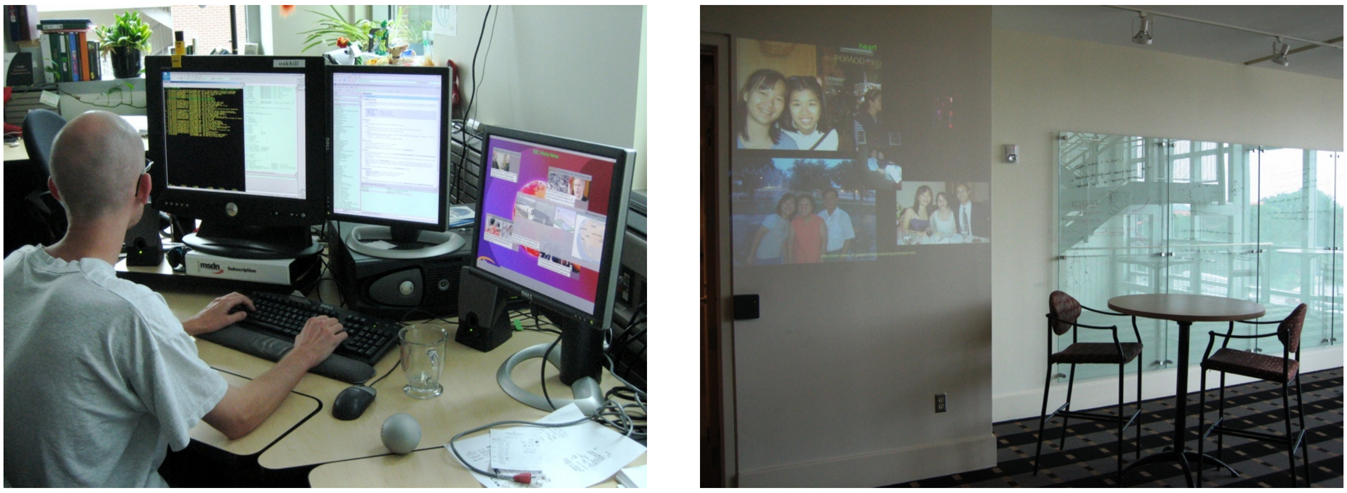


Figure 2. Cocoa Buzz running (a) on a secondary display on the desktop and (b) in a public lounge.

guide the design and construction of this application, we first interviewed current and potential users of an earlier non-customizable version of our software in order to identify the sorts of content they would like to see and the sorts of customizations they might wish to make.

Using these interview data, we defined a general-purpose multiple layer mechanism for data gathering and presentation. By coupling various data extraction heuristics with a mixture of general-purpose and specialized data gathering modules, users can create information awareness collages using a variety of data sources and formats in various presentation styles. In a preliminary study, we have found that some users do make use of these capabilities.

Cocoa Buzz runs on a secondary display on the user's desktop or on a large, shared display (see Figure 2). Approximately every minute, it presents a different collage, automatically generated from some particular source. Users can subscribe to various channels, each of which handles gathering data from the content publishers and generating an appropriate representation of those data. Some existing channels include collaborative news from Digg (Figure 1), a collection of dossiers on people within the research organization (Figure 5), and a collection of webcams chosen by the user, in addition to the standard news, weather, and traffic (Figure 9).

Users can subscribe to channels in order to create their own personal channel lineup. Channels offer basic configuration properties, such as postal (zip) code for the weather channel or tags for a Flickr channel. Channels also offer more advanced general purpose configuration options that are not tied directly to a particular data source. One of these configuration methods provides some simple heuristics to do things like extract the largest image on a web page or to perform a [Yahoo] image search using the keywords extracted from an RSS entry title. By using this more advanced interface, some users could extend the behavior of an existing channel or define a new one. Finally, we also provide an API through which programmers can define new components to augment the capabilities of the customization interface. We describe this customization interface and a pilot study in more detail in a later section.

USER-CUSTOMIZABLE CONTENT

Cocoa Buzz's predecessor was an application similar to the What's Happening screensaver [25]. Each minute, it would display a different informative collage. The primary content of this screensaver was collages generated from a webcrawler on the College of Computing web server. Additionally, the system displayed the standard set of awareness information: news headlines, weather, traffic, and stock data.

In informal conversations with the users of this system, however, a common theme arose: users expressed a desire to be able to tailor the content that appeared to their own interests. For example, one user who took public transportation indicated that she would prefer not to see the traffic information. Another user indicated that he would like to see the weather where his parents lived.

Many awareness applications reflect this user desire for customizable content [23, 8, 11]. Supporting such customization in a general purpose awareness application, however, is a difficult process. Simple customizations, such as selecting which information sources to subscribe to, may not be particularly difficult to support, but more complex operations must necessarily provide richer options to the user. Such operations add complexity both for the system, which must accommodate such customizability, and for the user, who must understand the interface provided by software.

As a result, awareness systems typically offer at least one of two modes of customization. The first is intended for customizations by the user, while the second caters to a developer or a power user. For example, a developer can create a Konfabulator [3] widget using XML, HTML, and JavaScript, defining any sorts of behaviors expressible within the supporting framework. Relatively few users, however, would be capable and willing to write the necessary code to

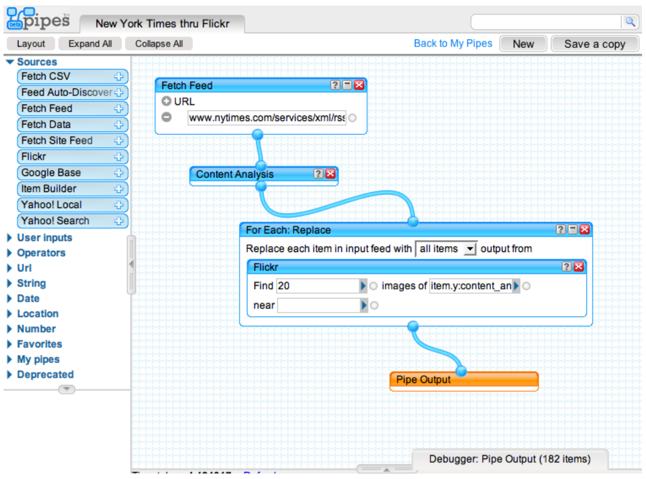


Figure 3. A Yahoo Pipe to mashup headlines from the New York Times with relevant photos from Flickr.

perform basic customizations of such a widget. Thus, the system also provides a framework whereby widget designers can expose basic properties for users to customize. Thus, the designer of a stock portfolio widget could perform the task-specific logic necessary in gathering stock market data and in representing those data to the user. The user, by contrast, would need only to enter the symbols of stocks in his or her portfolio, a comparatively straightforward operation.

Systems such as Buttons [16, 22], Konfabulator [3], Sideshow [8], Sidebar [17], Dashboard [6], and Google Gadgets [2] all use this bicameral approach to customization: developers create an object with specialized parameters and users perform basic configurations to those parameters. This dual-class approach, however, creates a significant hurdle for users who wish to transition from one role of interaction to another. If a user desires to modify a weather widget to show relative humidity, she must transition from providing simple properties via a configuration interface to writing code.

Various systems attempt to make it easier for users to write widgets by defining them in markup rather than in code [3, 17, 2, 6]. Although this approach does help lower the height of the barriers between these two approaches, it does not eliminate them completely. Users must still transition from one style of interaction to another.

Yahoo Pipes [4] attempts to blur this distinction between the two styles of interaction. Instead of requiring users to exit the program and write code or configuration files, Pipes integrates such behavior directly within the interface. The interface for each pipe prominently displays a button to view its source. In this way, as a user is configuring basic properties for a pipe, he or she can readily transition to editing the underlying behavior of the pipe. Furthermore, pipes are created using a visual programming interface in which users connect basic operators together via pipes (see Figure 3). This approach greatly reduces the barriers to creating such tools. Its focus, however, remains on programming. As such, pipes are still relatively complex for most users to create.

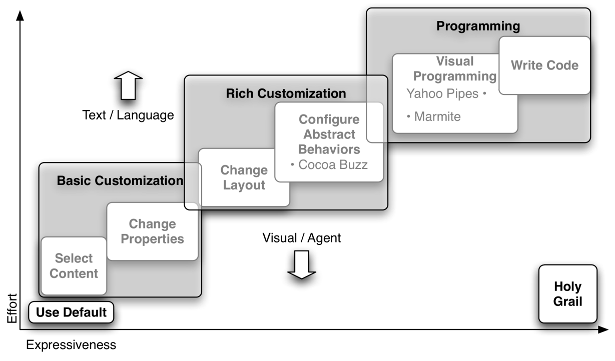


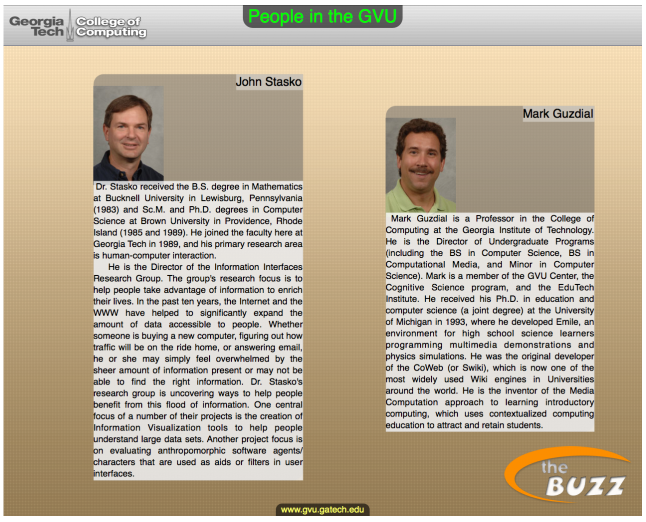
Figure 4. Range of customizations within an awareness application.

Marmite [24] attempts to reduce this complexity further by restricting the types of data flows possible and by providing hints as to possible operators in a data flow. Rather than providing a free-form visual programming interface as in Pipes, Marmite uses the concept of a dataflow and an interface inspired by Apple's Automator [5]. In this interface, the user starts out with an information source and defines a linear sequence of operations to perform on the data. At each step in the process, the system makes suggestions to the user about possible operators to use at that point. As a result, some users with spreadsheet experience were able to create interesting data mashups [24]. As with Pipes, Marmite still appears to be graspable only by users with at least spreadsheet programming abilities.

Both Pipes and Marmite provide a similar rich customization capability. Rather than writing code, users can create visual dataflows. These approaches reduce but do not eliminate the complexity involved in users creating their own dataflows. By providing an obvious transition between roles in the interface, a system can help to remove one potential barrier to rich customization. While it may become obvious how to customize the software in the interface, using that interface still remains necessarily complex. Thus, there is an inherent tradeoff between the expressiveness of a customization interface and its ease of use. As Figure 4 illustrates, user effort increases along with the expressiveness of the interface. Visual methods may help to reduce the effort, but can only offer a certain degree of benefit.

In this sense, there is a cliff [16] between the two styles of interaction, with tweaking properties of the data at the bottom and defining new behaviors at the top. Pipes and Marmite use two approaches to helping users climb this cliff. The first is to bring the top closer to the bottom by reducing the complexity involved in defining the new behaviors. The second is to build a ladder between the two levels in the form of obvious user interface transitions.

An alternate and complementary approach is to raise the bottom of the canyon by improving the expressiveness of the basic configuration interface. In order to avoid overburdening the user, however, a balance must be struck between the complexity and expressiveness involved in customizing the



John Stasko

Dr. Stasko received the B.S. degree in Mathematics at Bucknell University in Lewisburg, Pennsylvania (1983) and Sc.M. and Ph.D. degrees in Computer Science at Brown University in Providence, Rhode Island (1985 and 1989). He joined the faculty here at Georgia Tech in 1989, and his primary research area is human-computer interaction.

He is the Director of the Information Interfaces Research Group. The group's research focus is to help people take advantage of information to enrich their lives. In the past ten years, the Internet and the WWW have helped to significantly expand the amount of data accessible to people. Whether someone is buying a new computer, figuring out how traffic will be on the ride home, or answering email, he or she may simply feel overwhelmed by the sheer amount of information present or may not be able to find the right information. Dr. Stasko's research group is uncovering ways to help people benefit from this flood of information. One central focus of a number of their projects is the creation of Information Visualization tools to help people understand large data sets. Another project focus is on evaluating anthropomorphic software agents/characters that are used as aids or filters in user interfaces.

Mark Guzdial

Mark Guzdial is a Professor in the College of Computing at the Georgia Institute of Technology. He is the Director of Undergraduate Programs (including the BS in Computer Science, BS in Computational Media, and Minor in Computer Science). Mark is a member of the GVU Center, the Cognitive Science program, and the EduTech Institute. He received his Ph.D. in education and computer science (a joint degree) at the University of Michigan in 1993, where he developed Emile, an environment for high school science learners programming multimedia demonstrations and physics simulations. He was the original developer of the CoWeb (or Swiki), which is now one of the most widely used Wiki engines in Universities around the world. He is the inventor of the Media Computation approach to learning introductory computing, which uses contextualized computing education to attract and retain students.

Figure 5. The People in the GVU channel on Cocoa Buzz.

system. We have taken this gentle-slope approach [18] in designing a new information awareness system, called Cocoa Buzz.

Our goal in designing Cocoa Buzz is to focus on the middle customization space shown in Figure 4. As such, the Cocoa Buzz interface is not as powerful as those of Pipes and Marmite. In sacrificing the capability of the interface, we aim to make it easier for users to customize it while still retaining some degree of control over the behavior of the system.

REQUIREMENTS GATHERING

In order to understand what sorts of information users might wish to see and what sorts of customizations they might wish to make to the data, we interviewed twelve potential users of the software. Our goal was to understand both the kinds of information that users expressed a desire toward seeing in such an information awareness tool as well as their existing information practices.

We conducted interviews with twelve participants from the Georgia Tech community. All but two of these interviews were conducted at the participants' normal workplace. In the interviews, we asked users to describe their typical day and the sorts of information they routinely access. This included email, web sites, databases, *etc.*, as well as offline media such as television and radio. Our goal in this line of questioning was to identify what information sources might be important to support in a personal information awareness application.

We then asked users to walk us through their bookmarks and web browser histories. For privacy reasons, participants were given a chance to self-censor their bookmarks and web browser histories before our interviews. For each web site, participants described how they used that resource. What sorts of monitoring habits did they use? Did they tend to monitor a particular portion of a web page or web site, or did they monitor the web site as a whole?

Additionally, we asked what kinds of customizations users had made to their software. Did they change any of their preferences in any of their software applications? Did they use any mail handling rules in their email client? Had they customized a web portal page? Had they ever downloaded or used any software plugins or other types of hacks? In particular, we were interested in what experience our participants had in customizing their software and to what degree they were willing to do so.

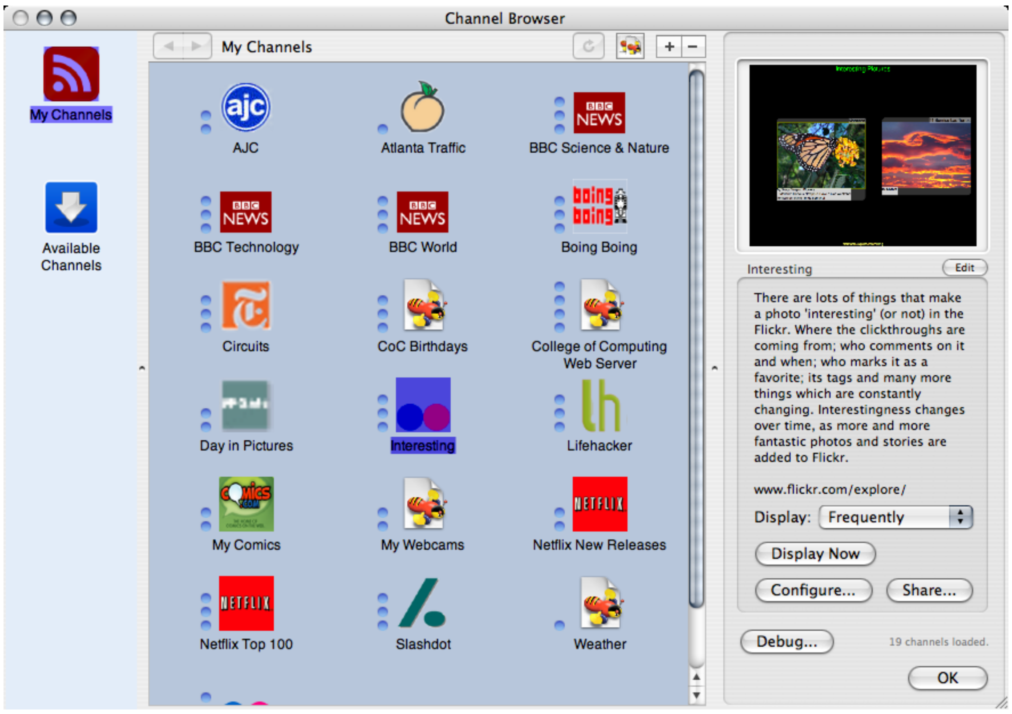
Finally, we gave our participants a stack of index cards with various potential data sources on them. We asked our users to rank each data source into three piles: those that they would particularly like to see in the awareness application; those that they might like to see; and those that they did not care about. For each source, they were then asked to describe why they cared about that particular source and what sorts of customizations they might wish to make to them (*e.g.* location for the weather).

These sources on the index cards included those used by the then-existing system along with sources that we as the designers of the system thought might be useful to include and sources that users had suggested in our pilot interviews. Finally, we also gave the participants a stack of blank index cards on which they could add their own.

Interview Results

Our participants consisted of members of the community for whom the original software, What's Happening, was intended to support. Within the College of Computing, we interviewed three Professors, seven graduate students, an administrative assistant, and an administrator, all working in Computer Science or related areas. Being a technical organization, we expect that our participants demonstrated a higher degree of capability and willingness to customize their software. We believe, however, that this is also similar to the demographic that is most likely to use a customizable information awareness application.

Interestingly, in our interviews, a quarter of our participants had used customizable portal pages and all but one of them had customized their portal pages. The participant who had not performed any customizations indicated that he had examined the options available but had chosen not to use them because they were too rudimentary. Furthermore, of these users who had customized their portal pages, none of them frequently visited those pages. These users had taken the time to customize the portal pages, but did not actually use the resulting artifact. This lack of use suggests that users may find portal pages too lacking in customization and/or presentation capabilities to be useful. For example, one user remarked that he had customized his page to figure out what he could do with the portal, but once he had done so, did not feel compelled to use it. Rather than condemning portal pages or information awareness applications in general, we see this behavior as potentially calling out the need for richer customization capabilities in information awareness applications. If users are unable to adequately customize their awareness tools, they will be unlikely to use them.



There are lots of things that make a photo 'interesting' (or not) in the Flickr. Where the clickthroughs are coming from; who comments on it and when; who marks it as a favorite; its tags and many more things which are constantly changing. Interestingness changes over time, as more and more fantastic photos and stories are added to Flickr.

www.flickr.com/explore/

Display: Frequently

19 channels loaded.

Figure 6. Browsing Cocoa Buzz channels.

Our primary goal in probing our users for their information monitoring habits, however, was to identify what sorts of triggers existed to their accessing particular pieces of information. In Kellar *et al*'s terminology for monitoring activities [13], we were looking at when participants tended to monitor data sources for browsing, fact finding, or information gathering purposes. Each of these different purposes suggests a different kind of interaction. For example, a user participating in a browsing activity may be content merely to sit back and let the system determine what content appears and when. For a fact finding activity, however, the user is motivated to see a particular piece of information at a particular time. As such, a direct path to that data should be available. Therefore, an application such as Cocoa Buzz, which provides a slideshow style of presentation, should provide a shortcut to an index of all of the available content if it is to support fact finding activities.

Finally, we used the index cards to provide an indication of what kinds of data sources our users might want to monitor. While the standard sources such as news headlines, traffic, weather, and stock quotes quickly rose to the top of the list, there were also some surprising sources identified. One user described a "pretty things" collection of pictures she had taken on her travels: "It sparks conversations and makes me happy."

Other sources tended to be relatively one-off in nature. For example, a professor wanted to see "what the snake says" on an introduction to programming class wiki. On the web page for the class in question, the professor puts quick class announcements in a speech bubble next to a cartoon Python snake. Seeing what the snake says would help him to keep in sync with the announcements he or any of the other teaching assistants might be making to the class. This particular example demonstrates an inherent limitation in awareness applications that focus on handling data in particular formats. Although data in semantic markup formats such as RSS is increasingly common, most content is more *ad hoc* in nature. For example, the speech bubble of the snake is a clever presentation trick made by the designer of the class web page. As a result, a general purpose heuristic to extract the meaningful content from a web page is unlikely to find "what the snake says" without some more specialized guidance. Instead, the "What the snake says" channel was implemented using a custom regular expression.

PROTOTYPE DESIGN GOALS

In our interviews, our participants identified a rich set of information sources that they accessed or would care to access on a routine basis. These information sources helped to guide the design of this interface by providing insight into the sorts of customizations users might wish to make. For example, eight of our twelve participants indicated that they desired to see interesting pictures from their photo collections, others' collections, or both. As such, one of the goals was to make sure that it was possible to create such a photo gallery using the customization capabilities of the system.

A further goal was to, wherever possible, use general purpose methods for gathering content and for presenting it. By using these general purpose methods, the particular information channels could more readily be extended to handle different kinds of data or presentation methods without resorting to writing specialized code. Nonetheless, there is an ease-of-use tradeoff between general purpose and specialized methods. Consider, for example, extracting images from the Flickr photo sharing web site. A general purpose approach might be to use a web crawler to extract the largest image from each page. But tuning that process to show only photos tagged with “Firenze”, for example, requires configuring the web crawler in certain precise ways that are unlikely to be intuitive to most end users.

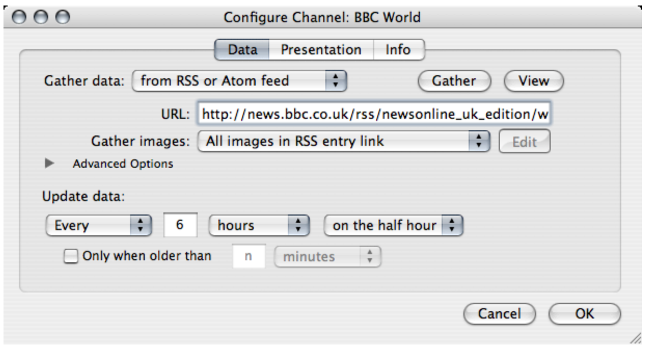
An alternative approach is to provide a specialized data gathering mechanism, or harvester, that lets the user configure task-relevant properties (in this case, photo tags) and handles the underlying extraction details behind the scenes. Such a specialized approach benefits by being able to provide the user with meaningful configuration options, using language and terminology from Flickr, which he or she is more likely to understand. However, by virtue of being specialized to the particular task they solve, these methods are inherently limited. They can only handle the particular data for which they were designed. As a result, they cannot be adapted to handle another sort of data or a different presentation style. In the preceding example, it would be impossible to configure the tool to use a different photo sharing service or to extract different data from Flickr without modifying the specialized components of the tool itself.

We have attempted to balance these tradeoffs in Cocoa Buzz by providing a combination of general purpose and specialized tools. Thus, in the preceding example, a user could configure a web crawler to gather all of the images from a particular set of web pages on the Flickr web site, or the user could specify a combination of tags, users, or groups (in Flickr’s terminology) from which to gather photos. The next section demonstrates some of this behavior.

USING COCOA BUZZ

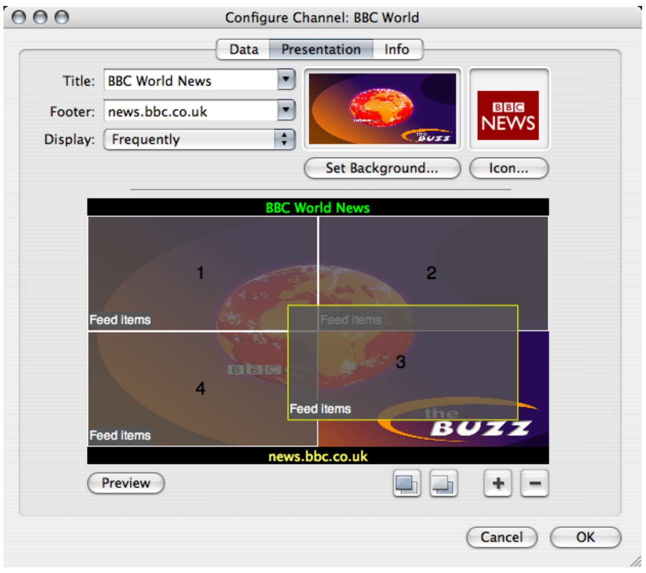
Whether Cocoa Buzz is running on an extra display on the desktop or on a large, public display, the machine’s owner can browse the running channels using the interface shown in Figure 6. In the center of the screen, Cocoa Buzz displays a list of all of the channels the user has subscribed to. On the right are details about the selected channel: a preview of the last collage shown, its name, and a brief description. Clicking the share button allows the user quickly to publish the channel for other users. Clicking the blue Available Channels button on the left of the screen allows the user to browse channels shared by other users of the software. In this way, users can navigate shared channels in the same way they navigate their own. We integrated this sharing mechanism directly into the interface because of the widely documented sharing behavior of users of customizable systems [14, 16]. We hope that by making sharing and browsing channels a basic operation in the interface, we can encourage the formation of a sharing culture.

To further encourage tinkering, users may double-click on a channel to view its configuration and can copy channels to easily create derivative behaviors. Figure 7 shows the channel configuration interface. Across the top of the screen are


Configure Channel: BBC World

Gather data: from RSS or Atom feed
URL: http://news.bbc.co.uk/rss/newsonline_uk_edition/w
Gather images: All images in RSS entry link
Update data: Every 6 hours on the half hour
Only when older than n minutes

Figure 7. Configuring the data gathering properties of a channel.


Configure Channel: BBC World

Title: BBC World News
Footer: news.bbc.co.uk
Display: Frequently

Figure 8. Customizing the layout of a channel.

three tabs controlling what data to show, how to show the data, and meta-information about the channel itself (such as its name or description). The first tab (shown in Figure 7) allows the user to select a data gathering method. In this case, an RSS feed of BBC world news headlines is used. A user can create a new RSS-driven channel simply by dragging an RSS feed onto the Cocoa Buzz display from, for example, a web browser, email client, or a feed reader.

In this particular example, the user could simply drag the RSS feed over from a web browser, press the OK button on the bottom of the screen, and a new channel would be created from the data in the RSS feed. The BBC, however, provides a rich RSS feed with well-written titles and descriptions of all stories, but does not include any images in the feed. As such, the resulting channel would be relatively text heavy and image bare. The collages generated by Cocoa Buzz are much more appealing and do a better job of conveying the information, however, when augmented with appropriate graphics. As a result, we have added an additional heuristic that proves useful for many such feeds: gathering all of the images found on the web page linked in each RSS entry. Thus, while an entry might not directly provide any images, it is likely to link to a web page of the full content which often does contain supplementary graphics. As such, a user may change the image gathering method in the interface from "All images in RSS entry" to "All images in RSS entry link," as shown in the Figure.

Clicking on the Presentation tab at the top of the window will bring up a layout template, shown in Figure 8. The top portion of the screen provides basic configuration options for the title and footer for the collage (which may be automatically provided from the underlying data source) and for the frequency with which each collage should be shown. The main portion of the screen allows the user to create and drag around various content regions. Each region represents a place on the screen where some particular piece of data will be drawn. The default template places five regions on the screen in a collage-like layout: four in each of the standard Cartesian quadrants with a fifth centered above the others. Figure 8 shows the user customizing a template by dragging a region into place.

By default, each region is bound to a random data entry provided by the data harvester. This default is appropriate for most data collections, where each entity is essentially a different instance of the same class (such as a news article or a photograph). Users can override these bindings by double-clicking on a region and associating the region with a different piece of the already-gathered data. This approach is more commonly used with heterogeneous data whose entities tend to remain the same but whose values themselves change. For example, the Georgia Navigator traffic website publishes the same set of maps, cameras, and sensors, but the values of those entities changes frequently (see Figure 9).

This template approach allows users to "paint" their own collages, brushing their data onto the screen. A user simply drags the various regions into the configuration he or she desires and associates each region with a piece of data. When it comes time to display the collage, the system inspects the data to identify its type and automatically determines an appropriate rendering method to depict it to the screen (the user can override this mechanism if she so desires). Thus, if the data to be drawn is numeric in nature, then a visualizer appropriate for depicting numeric data may be used. If the data has a title, images, and a description, then the system may use an RSS entry visualizer.

By using this approach, the various data harvesters can output multiple representations of the data they gather, allowing multiple visualization approaches to be used to depict them. For example, when the built-in weather harvester outputs the probability of precipitation, it outputs both an image ranging from clear to monsoon and a numeric percentage. Thus, a visualizer could use the appropriate stored value for its representation.

CUSTOMIZING DATA

The design of the data harvesting capabilities of Cocoa Buzz uses a multi-layer approach to data harvesting. Many users need not worry about the lower layers, however. The top layer, the harvester, focuses on getting data from the publisher. Existing harvesters include general purpose methods (such as a web crawler) and more specialized methods (such as a Flickr harvester built using the Flickr web service API). These harvesters can then delegate the content extraction to lower layer methods. This is the approach used by the RSS harvester in the earlier BBC World news example in which image gathering was delegated to a lower-level component, called a scraper, that extracts images from a web page.

We have taken great care in selecting the various scrapers available when customizing the data harvesters for a channel. The scrapers embody various heuristics that we have found are useful in creating collages for a variety of different web-based data formats. We could add many more heuristics or richer ways of creating complex, interconnecting components, as in Yahoo Pipes or Marmite. Doing so, however, would create additional complexity in the interface that we are trying to avoid[2]. Instead, the system provides a list of basic heuristics and a small handful of more complex ones, as follows:

All images with(out) captions on web page Extracts the URLs for all images in HTML `<img>` tags in a document, with or without their corresponding `alt` attributes.

Largest image with(out) caption on web page Extracts the URL for the largest image on a web page.

Image at web address Extracts a specified URL. This can be useful for content at fixed URLs, the contents of which are periodically updated.

All images in RSS entry Extracts the URLs for all images specified in an RSS entry.

All images in RSS entry link Extracts the URLs for all images embedded in the web page specified by an RSS entry's link.

From RSS entry link A mechanism that allows the application of another heuristic to the URL specified in the RSS entry's link. The "All images in RSS entry link" is a shortcut for this heuristic combined with "All images on web page"

From image search using RSS entry title Performs a keyword extraction on the title of an RSS entry and uses these keywords on a [Yahoo] image search.

Using a pattern matcher Applies a regular expression to return whatever text, if any, matches.

Most of these data extraction heuristics perform relatively straight-forward operations. There are, however, two methods that provide for some additional complexity. The "From RSS entry link" scraper allows an arbitrary scraper to be

[2]Cocoa Buzz does, however, expose a plugin architecture whereby programmers can extend the available harvesters, scrapers, and visualizers. Thus, if a programmer had code to identify "blueish" pictures, she could add a new "blueish pictures scraper" with relatively little effort.

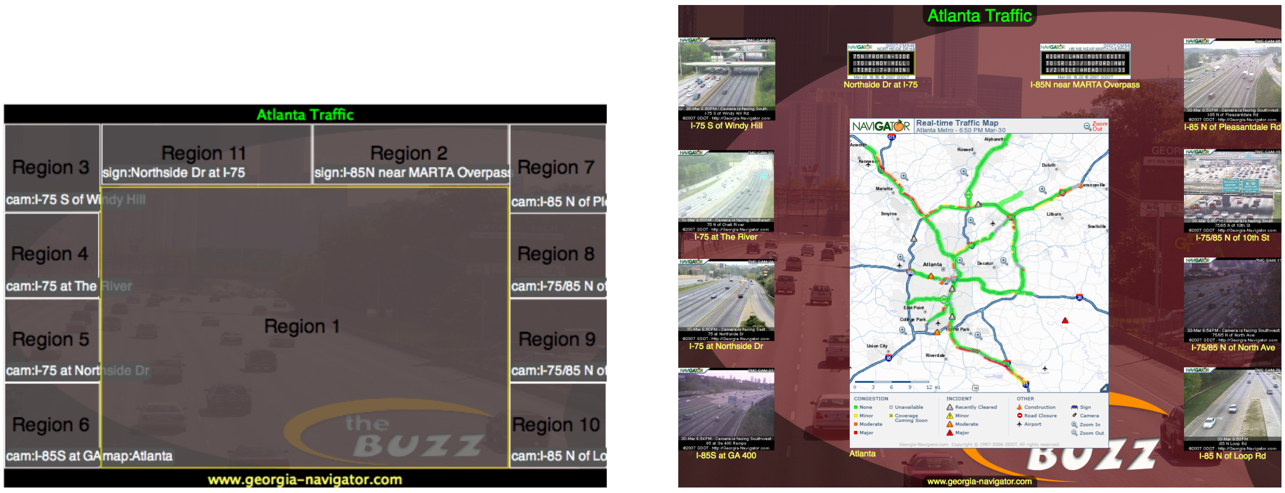


Figure 9. The layout template (left) for the Traffic channel (right).

used on the URL of an RSS entry (in fact, the "All images in RSS entry link" scraper is essentially syntactic sugar combining this scraper with the "All images on web page" scraper). In this way, some complex chaining of operations is possible, but within a highly constrained context. Our goal is to remove complexity that might overburden the user by constraining the choices possible. Thus, while the user can couple multiple methods together, he or she can only do so within a structured framework.

The other complex scraper uses a pattern matcher to find content. Although patterns are difficult to learn and to write, we provide a pattern matching mechanism within a relatively fine-grained context. Much in the same way that spreadsheets allow users to enter complex formulae within the rigid context of their document [19], Cocoa Buzz users enter patterns only within a narrow context. In this way, users do not need to write complex programs in code or using a visual programming interface. Instead, they supply a pattern within an isolated context. We hope that narrowing the context in this way might enable users to create these complex customizations without becoming overburdened.

Fortunately, writing patterns is an exceptional case intended only for when the other methods are insufficient. To aid the user in writing these patterns when they do become necessary, we provide a pattern editor interface, shown in Figure 10. The top region of the editor shows (an instance of) the underlying HTML to which the pattern will be applied. The bottom region is where the user composes his or her pattern. As the user enters the pattern, the display updates to show all matches to the pattern in the document. Additionally, any errors in the pattern expression are shown in grey in the space directly beneath the pattern. In this way, the user can get immediate feedback as he or she composes the pattern, helping to reduce the likelihood of errors.

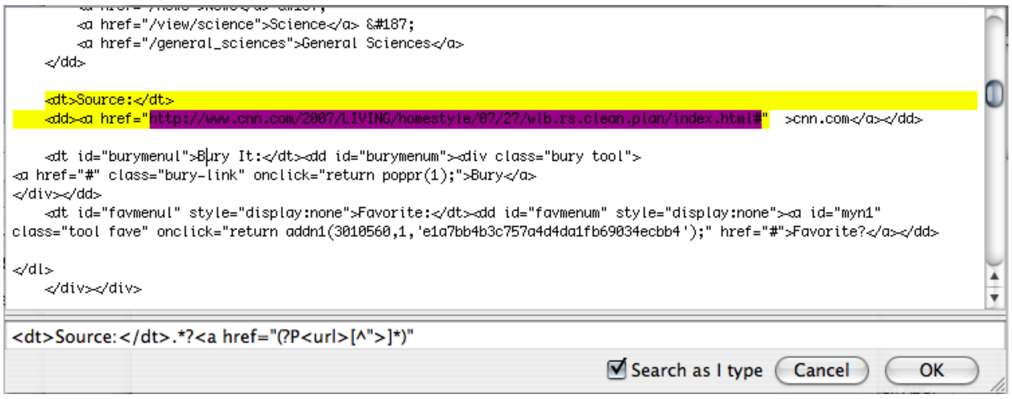
Content Extraction Heuristics

The heuristics embodied in the various scrapers are tailored specifically towards the sorts of data sources that our users identified in our interviews. In practice, they seem to work well for these tasks and for others that our users did not specifically request. Nonetheless, there are still many data sources for which these heuristics are insufficient or too complex. In particular, those data sources for which there is not a standard markup for their textual content. In order to handle these data sources, the user must either ignore any textual content, extracting just images from the underlying data, or she must write a complex pattern matching expression to define an extraction template. Such pattern expressions are useful when they are needed, but those cases should be minimized as much as is feasible.

There are, however, several viable alternatives to using regular expressions. The first is to consider ways to simplify the pattern expression language to something better suited to the specific task of extracting content from HTML and XML formatted documents. For example, all of the existing patterns that have been used in creating Cocoa Buzz channels extract selected elements or attributes within the source document. A simplified expression syntax, such as XPath queries [9], could help reduce the complexity users face when composing such patterns. Nonetheless, patterns will continue to remain complex.

Sifter [12] and Dapper [1] provide a method by which a user creates an extraction template from rendered web pages. In Sifter, the user selects the rendered desired entities of a web page, and the system infers the underlying HTML corresponding to the graphical elements the user selected. In this way, the user can create a data extraction template from rendered output. Dapper uses a similar approach but attempts to make the templates more robust by requiring multiple instances of the underlying data for its inferencing mechanism. Dontcheva's summaries framework [10] extends these approaches to allow the user to define extraction patterns, relationships between the extracted data, and to manage them.

If, however, no semantic understanding of the underlying data is required, and it is feasible to simply extract the rele-



```
        <a href="/view/science">Science</a> &#187;
        <a href="/general_sciences">General Sciences</a>
    </dd>

    <dt>Source:</dt>
    <dd><a href="http://www.cnn.com/2007/LIVING/homestyle/07/27/wlb.rs.clean.plan/index.html#"  >cnn.com</a></dd>

    <dt id="burymenul">Bury It:</dt><dd id="burymenum"><div class="bury tool">
<a href="#" class="bury-link" onclick="return poppr(1);">Bury</a>
</div></dd>
    <dt id="favmenul" style="display:none">Favorite:</dt><dd id="favmenum" style="display:none"><a id="myn1"
class="tool fave" onclick="return addn1(3010560,1,'e1a7bb4b3c757a4d4da1fb69034ecbb4');" href="#">Favorite?</a></dd>

</dl>
    </div></div>
```

```
<dt>Source:</dt>.*?<a href="(?P<url>[^">]*)"
```

Figure 10. Editing a pattern in Cocoa Buzz to extract the source URL for a story on Digg.

vant data and present it to the user as-is, the approach from Greenberg and Boyle's Notification Engine [11] could provide a useful alternative. The Notification Engine operates on the surface representation of the data rather than the underlying data themselves. Instead of inferring the underlying HTML corresponding to rendered output on the screen, the system extracts the rendered pixels. To monitor for changes, the system uses a landmark-based visual differencing engine to identify regions of web pages that have changed. If a change is detected in a monitored portion of a web page, that change is output as a graphical notification. This approach works particularly well in many cases and allows a certain degree of data format independence, but it is fragile to certain kinds of web page presentation styles and does not provide any semantically useful data. For example, it is difficult to compute derived data or to rewrap extracted text.

Regardless of the individual limitations of each of these approaches, they all offer beneficial heuristics for content extraction. By combining these approaches, a system such as Cocoa Buzz could handle many data sources and presentations with a relatively low user complexity.

DEPLOYMENT

To test how well these interfaces support users at creating these custom information awareness channels, we have performed a pilot deployment of the software within our organization. Six users, some of whom participated in the earlier interviews, ran the software on their primary work computers for about two months. Of the six users, all performed at least basic customizations to the system such as subscribing to or unsubscribing from channels. Additionally, five of the six users created derivative channels by modifying the configurations of existing channels. Most of these configuration changes involved editing basic channel properties, such as the tags used in a Flickr query or the URL of an RSS feed. Two users, however, created more complex customizations. For example, one user created a new channel from scratch using the web crawler to traverse his personal website and to create a collage of his wedding photos. Additionally, one user created a channel for his family vacation photos and shared it with other users of the system.

The system is also running on a large public display in a busy hallway. People walking by can glance at the screen and glean whatever happens to be showing at the time. The system focuses on community-relevant content, including content submitted by users. One channel, for example, shows photos with a particular tag (4thebuzz) on Flickr. Anyone can add their photos to the display simply by adding this tag to their photos. Using this mechanism, several members of the community have uploaded their conference pictures, photos of group meetings, and even user interface glitches.

So far, we have focused on whether users are able to understand and take advantage of the customization interfaces provided. Preliminary observations suggest that our technical users are at least able to make some use of the interfaces, but more data is needed to make concrete claims.

CONCLUSIONS AND FUTURE WORK

We have demonstrated a publisher-to-presentation information awareness application that offers promise at enabling end users to richly tailor their awareness environment. By focusing our attention on the space between basic customization and advanced visual programming, we aim to provide a stepping stone to help end users create novel behaviors without becoming overburdened by programming oriented concepts. Toward that end, we provide an abstraction of the data gathering process that affords such tailorability, and we have promising preliminary evidence to suggest that users may be able to take advantage of these capabilities.

More work, however, needs to done to examine more closely how effectively this approach supports users in performing real tasks. We plan to further refine the various heuristics embodied within the Cocoa Buzz interface and to conduct task-oriented usability studies to examine the efficacy of the interface. Additionally, we want to perform a richer analysis of real-world usage of the customization system. Are there particular triggers that encourage users to customize their interface and which we should more directly support? What barriers discourage such customizations and are there technological solutions to reduce those barriers? Finally, we would like to examine whether these triggers and barriers differ from those identified in other contexts [15].

REFERENCES

1. Dapper. Retrieved June 13, 2007 from `http://dappit.com`.
2. Google gadgets. Retrieved August 6, 2007 from `http://www.google.com/apis/gadgets/`.
3. Yahoo! Konfabulator. Retrieved June 13, 2007 from `http://widgets.yahoo.com`.
4. Yahoo! Pipes. Retrieved June 13, 2007 from `http://pipes.yahoo.com/`.
5. Apple, Inc. Automator, 2005. Retrieved June 13, 2007 from `http://www.apple.com/macosx/features/automator/`.
6. Apple, Inc. Developing dashboard widgets. Retrieved June 13, 2007 from `http://developer.apple.com/macosx/dashboard.html`.
7. Brian P. Bailey, Joseph A. Konstan, and John V. Carlis. The effects of interruptions on task performance, annoyance, and anxiety in the user interface. In *Proceedings of INTERACT 2001*, 2001.
8. J. J. Cadiz, Gina Venolia, Gavin Jancke, and Anoop Gupta. Designing and deploying an information awareness interface. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 314–323. ACM Press, 2002.
9. James Clark and Steven DeRose. XPath (XML Path Language). W3C Recommendation, World Wide Web Consortium (W3C), November 1999. Retrieved June 13, 2007 from `http://www.w3.org/TR/xpath`.
10. Mira Dontcheva, Steven M. Drucker, David Salesin, and Michael F. Cohen. Relations, cards, and search templates: user-guided web data integration and layout. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 61–70, New York, NY, USA, 2007. ACM.
11. Saul Greenberg and Michael Boyle. Generating custom notification histories by tracking visual differences between web page visits. In *GI '06: Proceedings of the 2006 conference on Graphics interface*, pages 227–234, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.
12. David F. Huynh, Robert C. Miller, and David R. Karger. Enabling web browsers to augment web sites' filtering and sorting functionalities. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 125–134, New York, NY, USA, 2006. ACM Press.
13. Melanie Kellar, Carolyn Watters, and Kori M. Inkpen. An exploration of web-based monitoring: implications for design. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 377–386, New York, NY, USA, 2007. ACM Press.
14. Wendy E. Mackay. Patterns of sharing customizable software. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 209–221, New York, NY, USA, 1990. ACM Press.
15. Wendy E. Mackay. Triggers and barriers to customizing software. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 153–160. ACM Press, 1991.
16. Allan MacLean, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. User-tailorable systems: pressing the issues with buttons. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 175–182, New York, NY, USA, 1990. ACM Press.
17. Microsoft. Vista Sidebar, 2007. Retrieved August 30, 2007 from `http://www.microsoft.com/windows/products/windowsvista/features/details/sidebargadgets.mspx`.
18. Brad A. Myers, David Canfield Smith, and Bruce Horn. Report of the "End-User Programming" working group. *Languages for Developing User Interfaces*, pages 343–366, 1992.
19. Bonnie A. Nardi. *A small matter of programming: perspectives on end user computing*. MIT Press, 1993.
20. Zachary Pousman and John Stasko. A taxonomy of ambient information systems: four patterns of design. In *AVI '06: Proceedings of the working conference on Advanced visual interfaces*, pages 67–74, New York, NY, USA, 2006. ACM Press.
21. Paul Rademacher. Housing maps, 2005. Retrieved March 27, 2007 from `http://www.housingmaps.com/`.
22. George G. Robertson, Jr. D. Austin Henderson, and Stuart K. Card. Buttons as first class objects on an X desktop. In *Proceedings of the 4th annual ACM symposium on User interface software and technology*, pages 35–44, New York, NY, USA, 1991. ACM Press.
23. John Stasko, Todd Miller, Zachary Pousman, Christopher Plaue, and Osman Ullah. Personalized peripheral information awareness through Information Art. In *Proceedings of UbiComp '04*, pages 18–35, Nottingham, U.K., September 2004.
24. Jeffrey Wong and Jason I. Hong. Making mashups with Marmite: towards end-user programming for the web. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1435–1444, New York, NY, USA, 2007. ACM Press.
25. Q. Alex Zhao and John Stasko. What's Happening?: Promoting community awareness through opportunistic, peripheral interfaces. In *Proceedings of the Advanced Visual Interfaces Conference*, pages 69–74, Trento, Italy, May 2002.