Orthogonal Extensions to the WWW User Interface Using Client-Side Technologies.

Armando Fox, Steven D. Gribble, Yatin Chawathe, Anthony S. Polito,
Andrew Huang, Benjamin Ling, and Eric A. Brewer

1 The World-Wide Web as a Universal User Interface

Our work is motivated by three trends. First, the ubiquitous migration of services to the World Wide Web is due in part to its simple, consistent, and now universal user interface: navigation by following links and filling out HTML forms are interactions familiar to even novice Internet users. Second, client-side extension technologies such as Java and JavaScript allow sites to extend and "personalize" the behaviors and interfaces of their services, with portable user-interface elements that integrate transparently into the browser's existing interface.

Finally, there has been a recent surge of interest in *proxy-mediated* access to the Web, in which proxy agents in the network infrastructure provide caching [1], anonymize user requests [2], or accelerate Web access via datatype-specific lossy compression [3, 4, 5]. Recent results show that these services can be built scalably and cost-effectively, and can shield the user from the limitations of their Internet connections or client platforms. Not surprisingly, the services have become increasingly powerful and therefore parameterizable and customizable by each user, resulting in increased attention on the design and implementation of the user interface by which the service can be controlled [8]

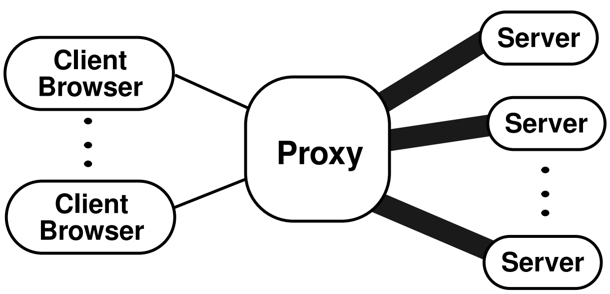

Figure 1: A Proxied Web Architecture

2 Orthogonal Extensions to the Web's UI

We describe our experience implementing *orthogonal extensions* to the existing WWW user interface, to support user control of intelligent services. Our extensions are orthogonal in that they provide an interface to a *service*, which complements the Web browsing experience but is independent of the content of any particular site. We base our experiments on the TranSend service at UC Berkeley, which performs lossy compression on inline images to accelerate dialup Web access for a community of 25,000 subscribers. The service keeps a separate "preferences profile" for each user, which allows each user to vary the aggressiveness of lossy compression, selectively turn off the service for certain pages, and select the type of interface provided for *refinement* of degraded (lossily compressed) content.

We are exploring three technologies for implementing the TranSend service interface: HTML decoration, Java, and JavaScript. The accompanying video demonstrates prototypes of all three mechanisms.

We now briefly describe each approach.

2.1 HTML Decoration

Most previous attempts to provide such an orthogonal service interface have relied on *HTML decoration*, i.e., inserting elements into the original HTML on the fly before passing the HTML to the client. (Various Web sites [6] similarly use *dynamic HTML* to vary the site's page composition, to exploit browser-specific features such as frames and imagemaps.) Our earliest efforts to provide an interface to TranSend were based on HTML decoration. In particular:

- We insert HTML links alongside each (lossily compressed) inline image. Following such a *refinement link* causes the original image to be retrieved.
- We insert a distinctive TranSend icon at the top of each visited page. The icon serves to remind the user that she is viewing a page filtered through the TranSend service. Also, clicking on the icon takes the user to a *control panel* where she can enable and disable the service and vary the aggressiveness of distillation. The control panel is just an HTML form with appropriate controls for each setting.

Figure 2: Elements of the HTML form interface for controlling the proxy distillation aggressiveness and the HTML decorations inserted on each page.

2.2 Java Dashboard

Our second prototype interface replaces the TranSend icon with a *dashboard* similar to those found in desktop applications. Currently the dashboard allows access to all of the same settings as the HTML form. TranSend inserts HTML elements necessary to display the dashboard, which is a Java applet, at the top of each visited page. If the user finds the dashboard visually obtrusive, it can be "minimized" into an icon and maximized again later if the user wishes to change settings.

Figure 3: The Java "Dashboard" that is inserted at the top of each page served to the user through the proxy.

2.3 JavaScript

Both of the above mechanisms suffer from a clumsy interface for image refinement: when the user clicks a refinement link (added to the HTML), the original image appears on its own blank page. We turned to JavaScript to remedy this problem:

- We annotate each visited page with a short JavaScript program which, when activated, causes each of the inline images on the page to be reloaded *in place* and replaced with the original versions;
- We add an additional icon labeled "Refine All" to the top of each page, which activates the JavaScript program.

3 Technical Issues

We encountered a number of interesting challenges in the implementation of these interfaces, for example:

- If users disable TranSend (i.e. turn off proxies) and attempt to view client-cached pages, the HTML decorations will no longer work. This is a specific instance of a general class of problems already experienced by caching proxies.
- Because our SGML scanner is less tolerant of ill-formed HTML than most commercial browsers, we occasionally introduce HTML elements that change the behavior or appearance of the page significantly.
- Keeping client-side state (e.g. which quality-level button appears selected in the dashboard) consistent with service-side state (how aggressively to compress) is awkward with the stateless HTTP protocol, which has no notion of "session state".

In addition, some users may refuse to enable Java or JavaScript because of the associated security risks [7], or may not have browsers that support these languages (JavaScript in particular is a moving target).

4 User Interface Discussion

As systems researchers, the more interesting (and less expected) challenges were user-interface challenges. Some of the ones that generated the most discussion included:

- "Collisions" with browsers' built-in UI: Java and JavaScript do not recognize "nonstandard" mouse events such as "shift-click" or "right-button click". JavaScript in particular leaves no option but to overload the left-click behavior, limiting the extent to which it may be used to extend the client interface.
- HTML decoration may adversely affect pages with precise pixel-level layout, fixed-size non-scrollable frame areas, etc. Doing the "Right Thing" for every page would basically require reproduction of the browser's rendering behavior at the service. Although we are trying to decrease our reliance on HTML decoration, we expect it to remain a popular option for security-conscious users or users with older browsers.
- How intrusive or effective do users perceive the different styles of interfaces to be? How many controls can be visible in a dashboard-style interface before its complexity overwhelms the user, defeating the appealing simplicity of the WWW interface? Should the dashboard be "customizable", similar to (e.g.) the Microsoft Office toolbars?
- The HTML-decoration interface is currently deployed in the production TranSend service; the Java and JavaScript experimental interfaces are expected to be deployed soon, and users will be able to select from among the different interfaces. Our ability to track which users are using each interface, and how often they switch, will form the basis of an interesting HCI experiment.

5 Conclusions

Proxied Internet access, through services such as TranSend, customizes content from existing servers according to each user's preferences and specifications. Orthogonal user-interface extensions complement this model by enabling users to customize the way they view and interact with the content, while remaining compatible with existing commodity browsers. We believe that this problem domain is a natural fit for technologies such as Java and JavaScript, and expect their impact in this area to equal or exceed their importance in providing site-specific services and interfaces.

We encourage the reader to try TranSend (and send feedback) by visiting: `http://transend.cs.berkeley.edu`

6 References

[1] C.M. Bowman et al. Harvest: *A Scalable, Customizable Discovery and Access System*. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, August 1994

[2] Community Connexion, Inc. *The Anonymizer.* http://www.anonymizer.com

[3] A. Fox and E. A. Brewer. *Reducing WWW Latency and Bandwidth Requirements via Real-Time Distillation.* Proc. WWW-5, Paris, May 1996.

[4] A. Fox, S. D. Gribble, E. Brewer and E. Amir. *Adapting to Network and Client Variation Via On-Demand Dynamic Distillation.* Proceedings of ASPLOS-VII, Boston, October 1996.

[5] A. Fox, S. D. Gribble, Y. Chawathe, and E. Brewer. *The TranSend Proxy Service*. http://transend.cs.berkeley.edu.

[6] Inktomi Corporation: *The Inktomi Technology Behind HotBot*. May 1996. http://www.inktomi.com/whitepap.html.

[7] Princeton University Secure Internet Programming, http://www.cs.princeton.edu/sip/.

[8] Takao Shimada et al. *Interactive Scaling Control Mechanism for World-Wide Web Systems*. Proceedings WWW-6, Santa Clara, May 1997.