Remixing the Web: Tailoring Applications using Programmable Proxies inside Web Browsers

Leslie Wu, Joel Brandt, Scott Klemmer
Stanford University HCI Group
Computer Science Department
Stanford, CA 94305, USA
[lwu2, jbrandt, srk]@cs.stanford.edu

ABSTRACT
This note reports on the motivation for and design of an infrastructure for presenting tailored web applications as services. We conducted a diary study of mobile information needs, finding that a significant majority of participants' desired information was *available* on the web, just not in a mobile-friendly format. This suggests there is latent value in lightweight tools that tailor web applications for mobile use. Browser extensions have emerged as perhaps the most lightweight and intuitive method for enabling end-users to tailor web applications, likely because browser-side approaches work fluidly with the logged-in web and because it most effectively leverages a diverse ecology of existing web development tools. However, client-side extensions are, well, client-side — inhibiting their portability, especially to the stripped-down browsers common to the mobile web. This note introduces *re:mix*, an architecture that delivers both the development benefits of browser-based application tailoring and the server-side benefits of proxy-based rewriting.

Author Keywords
Software tailoring, mobile web, end-user programming

ACM Classification Keywords
H.5.2. [Information interfaces & presentation]: User Interfaces—*Graphical user interfaces*. H.5.4. [Information interfaces & presentation]: Hypertext/Hypermedia—*Architectures*

INTRODUCTION
As more software applications are deployed through the Web and the diversity of web-enabled devices increases, there is significant value in support users' customization and tailoring of existing web applications. Current tools such as the Greasemonkey browser extension have gained significant traction—more than 8,000 Greasemonkey scripts are available from the web site *userscripts.org* [3]. Unfortunately, these existing tools break the service nature of web applications because their runtime web page rewriting occurs browser-side. Conversely, server-side tools such as programmable proxies allow the service nature of applications to be retained, but are more difficult to program because they lack the closeness of mapping present in browser-side tools.

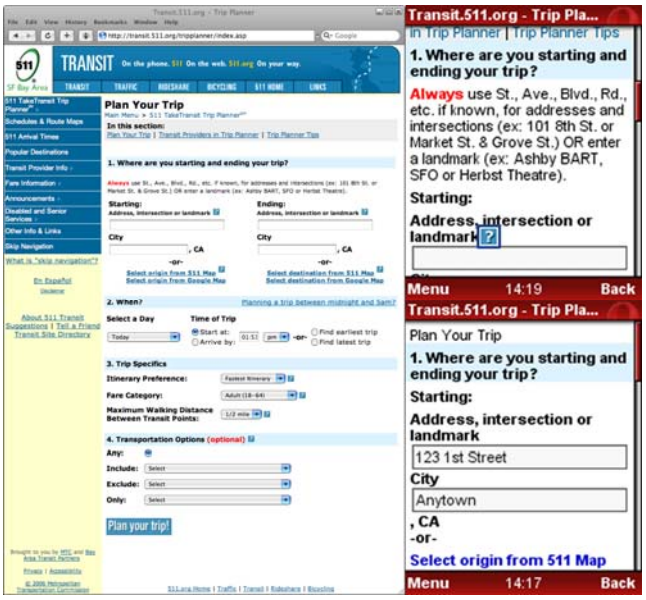

Figure 1. When a desktop web page (left) is automatically transcoded (top right), the result often requires excessive scrolling. An application tailored with re:mix (bottom right) can be more concise and support automation, such as form pre-filling.

Several factors affecting software tailoring have changed since early computer systems began supporting user-driven customizations. First, software designers increasingly develop their user interfaces and make their data accessible in standardized markup languages (such as HTML or XML). Second, software applications are increasingly deployed as a "software as a service," meaning that they are centrally managed and network-based [7]. The combination of these two factors—ubiquitous markup and always-on services—has enabled lead users to recombine elements from existing

applications and services in novel ways, such as in data mash-ups [10, 15].

This work was motivated by the observation that the web is available—but not necessarily usable—in an increasing diversity of situations through mobile computing devices. We began with a hypothesis that both *a large majority of users' mobile information access needs go unmet*, and that *a large majority of these latent needs could be satisfied by tailoring existing web applications.* In this note we report on a need-finding study that confirms this hypothesis, review existing browser-side and server-side tools for tailoring, and present a new tailoring architecture called re:mix that combines the benefits of both types of tools.

MOBILE INFORMATION ACCESS NEED-FINDING

To validate this hypothesis, we conducted a two-week long diary study about mobile information access needs. The study comprised 23 participants recruited from our university campus (14 female, 9 male) with a median age of 20 (range 19 – 28). Participants were asked to complete a structured diary entry about each information access need that arose while they were mobile, regardless of whether the need was met. In addition to the need itself, participants were asked to record contextual information about the situation in which the need arose—what time it was, where they were, and who they were with. They were asked to submit, on average, at least two needs per day (although this was not enforced). Participants were asked to err on the side of "submitting too much" when assessing whether their need was indeed an "information access" need.

In all, 442 entries were submitted. The second author pared these entries down to 190 to remove needs that were not information access needs (*e.g.* entries such as "I wish my phone didn't crash"). Of these, participants stated that 159 (85%) went unmet. The needs were then independently coded by two researchers (one of whom was external to the project) with respect to whether it was possible to meet the need today using existing web applications and a modern desktop web browser. Possible codings were: "Yes: can meet need today using desktop web", "Almost: could quickly build desktop website to meet need out of existing websites", "With Support: could quickly build desktop website to meet need if appropriate infrastructure services existed.", and "No: meeting this need would require significant development and/or innovation."

Both coders reported that the majority, 66%, of mobile information access needs could be completely met by the current desktop-based web (cross-coder correlation of 0.66). Coder 1 reported that an additional 7% could "almost" be met, and an additional 14% could be met "with support". Coder 2 reported 4% and 14% respectively (with cross-coder correlations of .40 and .26 respectively). Summing these values, the coders reported that an average of 82% of the needs could be met with little or no development or innovation (cross-coder correlation=0.42).

An informal survey of the diary entries indicates that both the information access needs and the contexts in which they arise are quite diverse. Multiple customizations of a single web application will often be necessary to satisfy needs that arise in different contexts. We suggest that this has two implications for tool support. First, tools should allow a diversity of tailored design variants to exist in parallel, and allow users to build upon existing customizations. Second, tools should target a broad community of lead users rather than the service-providing organizations themselves in order to harness the labor necessary to create this diversity.

REMIXING THE WEB

The reported needs suggest that tailoring existing applications typically involves two classes of tasks: making user interface modifications (*e.g.* reducing visual clutter, or automating a commonly-performed behavior such as logging in), and adding functionality by *mixing in* data from infrastructure services (*e.g.* pre-filling a form with current location data). We introduce the term *remixing* to refer to tailoring that involves these classes of modification. To make this notion more concrete, we introduce a use case drawn from our need-finding.

Use Case: Remixing Mobile Wayfinding

Elaine is a performer who makes frequent trips to different venues. She sometimes makes use of Google Maps to find directions, but often prefers to plan her trip on 511.org, a service that consolidates public transportation information.

Elaine navigates to 511.org on her mobile device and selects a bookmarklet (a bookmark that executes a JavaScript function) that redirects her to a list of available remixes for her current page. She chooses one that *mixes in* data from FireEagle [1], a web service that allows a user to share location information with other applications. This tailored application, is also accessible from any other web-enabled device. The design variant chosen includes some basic visual redesigns; it also automatically pre-fills her starting address, by programmatically querying the FireEagle web service (see Figure 1, bottom right).

While this improves the mobile wayfinding for Elaine, Elaine prefers to navigate using visual landmarks rather than with textual directions. When looking up directions online she opts to use a remixed version of Google Maps that mixes in photographs of landmarks from the Flickr web site, which offers a panoply of geotagged photographs. As she navigates turn by turn, she sees directions from Google and photography from Flickr, in one seamless user experience.

RELATED WORK

Prior work in web tailoring falls into two areas: browser extensions for client-side tailoring and programmable HTTP proxies that interpose between the web browser and web servers [4] (see Figure 2).

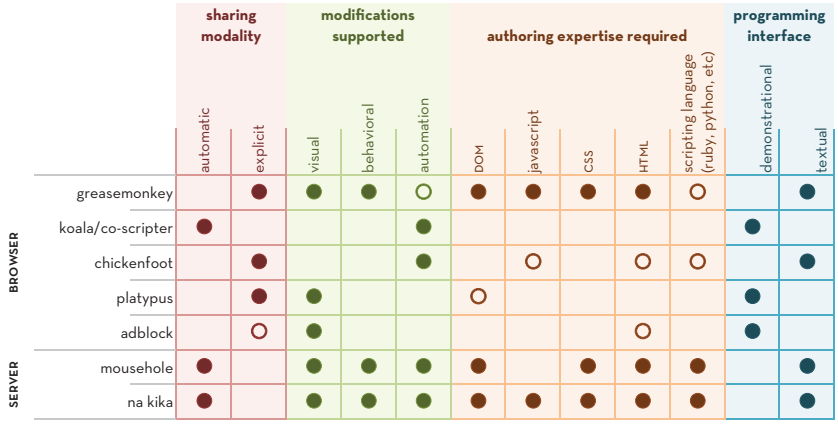

Figure 2. Existing tools for web tailoring. Many browser-based tools require minimal expertise, but make sharing of tailored applications difficult. Server-side tools are more powerful, but require a great deal of expertise to program.

Browser-side tools [5, 6, 12, 13], have the advantage of easily supporting the modern logged-in and AJAX-enabled web. Additionally, their location inside the browser allows for a close mapping between the definition language and the rendered result, which can lower the amount of expertise required. For example, Platypus [14] presents a direct-manipulation interface for visually redesigning pages. Browser-side tools often provide a mechanism for users to share their customization with others—the userscripts.org website for sharing Greasemonkey scripts is one such example. However, these customizations must typically be shared explicitly (*e.g.* by one user copying a script file to a server, and then another user downloading and installing it). Even when the sharing is made explicit—as is the case with Koala (now called Co-Scripter)—the customizations only work inside browsers which have the necessary extensions or features. This points to the primary disadvantage of current browser-based tools: tailored applications do not retain their original software-as-a-service nature. This means, for example, that they cannot be used to tailor web pages for mobile devices that have a closed architecture.

In contrast, programmable web proxies [2, 9] support tailored applications that retain their software-as-a-service architecture. Traditionally, however, these tools have a do not address the modern web because they do not execute JavaScript, nor robustly deal with session management. Finally, programmable proxies currently require a great deal of technical expertise to program because appropriate development and debugging tools have not matured.

A distinct but related form of software customization on the web is the *mash-up*, where two or more web services or data feeds are used as building blocks to create a new application. Because mash-up components provide data but not a user interface—the exception that proves the rule being the Google Maps API—mash-ups require their developers to create an interface.

ARCHITECTURE SUPPORT FOR TAILORING

As Bolin points out [4], supporting customizations within the browser environment allows the customization tool to access pages as the user sees them, affected by style sheets, session identifiers, and security restrictions. For the same reasons, our re:mix architecture implements a programmable proxy on top of the Firefox browser (see Figure 3). We have implemented the re:mix proxy inside POW[11], a Firefox extension that runs an HTTP server inside the browser.

At design time, users create remixes using the Firefox tensions of their choice. At runtime, *two* browsers are employed: a server-side browser kernel performs the rewriting, enabling any browser—even a lightweight one—to be the client. A web request proceeds as follows: First, the client browser requests a page from the proxy. The proxy loads the requested page inside a full-featured browser that has the appropriate extensions to perform the necessary remixing. After the page is fully loaded and the remixing is complete, the proxy transmits the resulting document object model (DOM) to the client.

Many mobile browsers and desktop browsers on public terminals do not allow the user to specify an HTTP proxy. In order to support such browsers, we use a URL-based approach similar to that employed by content-caching services such as CoralCDN [8]. Users simply request a remixed version of a page through a small modification to the URL. For example, a remixed version of 511.org might be available at http://10.0.0.1:8080/transit.511.org/tripplanner/, where "10.0.0.1:8080" is the IP address and port of the user's re:mix proxy, and the remainder of the URL is the URL to be remixed.

As discussed earlier, remixing often involves mixing in small bits of data from other services. Often times, the data to be mixed in may not be accessible by a cleanly-defined web API, nor can it easily be scraped, because it only exists on the logged-in web. In these situations, re:mix can be used *recursively* to access the desired information. This is perhaps made most clear with an example: a user may wish to mix her contacts' status information from a social network service into her webmail application. This status information is not available via an API, and only exists on the logged-in web. Because re:mix presents itself as a URL-based proxy, it *can be used recursively by the extensions that do the rewriting.* For example, the Greasemonkey script that rewrites the webmail interface can access the social network data by requesting a logged-in webpage through re:mix. Note that it would not be possible to access

this information using Greasemonkey alone. While Greasemonkey can modify a page on the logged-in web once it is loaded in the browser, it cannot *explicitly fetch* information from the logged-in web.

CONCLUSIONS AND FUTURE WORK

The application tailoring described in this paper was implemented without extensive tool support for interface and interaction redesign. Although the architecture presented enabled the production of tailored design variants, authoring these remixed applications is currently time consuming, and requires a certain level of technical web expertise. This suggests the need for better design tools built on top of the re:mix architecture. For example, a design tool for authoring remixes could integrate a demonstrational interface that allows lead users to combine automation and customization by demonstration. On a technical level, more work needs to be done in securing re:mix and in disclosing the nature of a re:mix script to a user. As it is, managing trust across web services is not easily done.

Finally, we believe that the open architecture described could enable tailoring of existing web applications for accessibility and universal access. Firefox is an accessible browser, but the application web at large is not universally accessible. The re:mix architecture may provide a way for proprietary assistive technology web clients, such as screen readers, to better integrate with existing proprietary applications, in such a way that the tailored, more accessible applications are still available as large-scale services.

We have motivated and presented re:mix, an architecture for tailoring web applications. This architecture brings together the benefits of two classes of existing tools. First, it enables browser-side authoring which allows for greater closeness of mapping in the development process. Second, it allows tailored applications to be provided as a service so they can be used on mobile and lightweight clients.

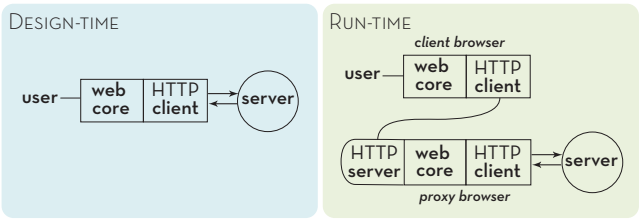

Figure 3. The re:mix architecture at design time and runtime.

REFERENCES

1 FireEagle, 2007. http://fireeagle.research.yahoo.com
2 MouseHole, 2007. http://code.whytheluckystiff.net/mouseHole
3 Userscripts.org, 2007. http://userscripts.org
4 Bolin, M., End-User Programming for the Web, Massachusetts Institute of Technology, Electrical Engineering and Computer Science, Cambridge, 2005.
5 Bolin, M., M. Webber, P. Rha, T. Wilson, and R. C. Miller. Automation and Customization of Rendered Web Pages. In ACM Symposium on User Interface Software and Technology. ACM Press, 2005.
6 Boodman, A., Greasemonkey, 2007. https://addons.mozilla.org/en-US/firefox/addon/748
7 Brewer, E. A. Lessons from Giant-Scale Services. Internet Computing, IEEE 5(4). pp. 46-55, 2001.
8 Freedman, M. J., E. Freudenthal, and D. Mazières. Democratizing Content Publication with Coral. In Proceedings of USENIX Symposium on Networked Systems Design and Implementation, 2004.
9 Grimm, R., G. Lichtman, et al. Na Kika: Secure Service Execution and Composition in an Open Edge-Side Computing Network. In Proceedings of USENIX Symposium on Networked Systems Design and Implementation. pp. 169-82, 2006.
10 Hartmann, B., L. Wu, K. Collins, and S. R. Klemmer. Programming by a Sample: Rapidly Creating Web Applications with d.mix. In Proceedings of UIST: ACM Symposium on User Interface Software and Technology. ACM Press, 2007.
11 Kellogg, D., POW — Plain Old Webserver, 2007. https://addons.mozilla.org/en-US/firefox/addon/3002
12 Little, G., T. A. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan. Koala: Capture, Share, Automate, Personalize Business Processes on the Web. In Proceedings of SIGCHI Conference on Human Factors in Computing Systems. ACM Press, 2007.
13 McDonald, M., Adblock, 2006. https://addons.mozilla.org/en-US/firefox/addon/10
14 Turner, S. R., Platypus, 2007. http://platypus.mozdev.org/
15 Wong, J. and J. I. Hong. Making Mashups with Marmite: Towards End-User Programming for the Web. In Proceedings of SIGCHI Conference on Human Factors in Computing Systems. ACM Press, 2007.