Remixing The Web: Enhancing Tailoring Using Programmable Proxies

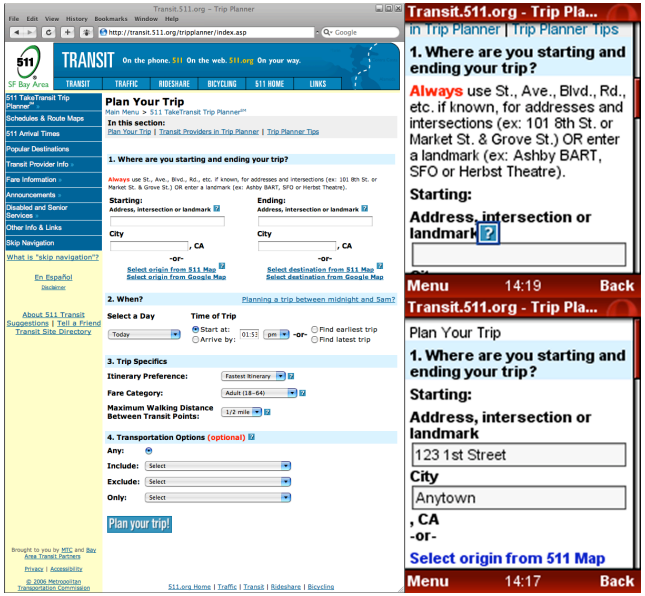

Figure 1. When a desktop web page (left) is automatically transcoded (top right), the result often requires excessive scrolling. An application tailored with re:mix (bottom right) can be more concise and support automation, such as form pre-filling.

Joel Brandt
Stanford HCI Group
Computer Science Dept.
Stanford, CA 94305 USA
jbrandt@cs.stanford.edu

Leslie Wu
Stanford HCI Group
Computer Science Dept.
Stanford, CA 94305 USA
lwu2@cs.stanford.edu

Scott R. Klemmer
Stanford HCI Group
Computer Science Dept.
Stanford, CA 94305 USA
srk@cs.stanford.edu

Abstract
This note introduces *re:mix*, an infrastructure for presenting tailored web applications as services. How can users most effectively tailor this web content for mobile consumption? Currently, there are two approaches to tailoring web interfaces: in-browser extensions and programmable proxies. Browser extensions facilitate direct-manipulation tailoring, leverage the massive browser-based developer ecology, and build upon a ubiquitous platform. Unfortunately, existing browser-based tailoring systems yield a client-side application, inhibiting their portability, especially to stripped-down mobile browsers. Programmable proxies enable the tailored application to be deployed as a service, but current systems lack the environmental benefits of the browser. This paper introduces a "best-of-both-worlds" approach to tailoring web pages through the insight that the browser platform can also serve tailored applications.

Introduction
There is significant and increasing value in supporting users' customization and tailoring of existing web applications. Current tools such as the Greasemonkey browser extension have gained significant traction. Unfortunately, these existing tools break the service nature of web applications crucial to the success of the mobile Internet: because their runtime web page rewriting occurs browser-side, the improvements a user makes to a

public transportation trip planner at home (Figure 1) will not be reflected on his or her office PC or mobile device. 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 and robust software libraries present in browser-side tools. Our work combines these approaches, introducing an architecture that allows the browser to both provide a platform for tailoring at design-time and serve tailored applications to other devices at run-time.

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. 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 [3].

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. This note reviews existing browser-side and server-side tools for tailoring, and present a new tailoring architecture called re:mix that combines the benefits of both approaches.

Remixing the Web

Before beginning this work, we conducted a need-finding study (not reported on here) on mobile information access needs. We found 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. Our example in Figure 1 involves both classes: the page is visually restructured to fit the constraints of a mobile device and the user's current location is "mixed in" to the form automatically.

A distinct but related approach to customizing web applications is mash-ups, where two or more services or data feeds are used as building blocks to create a new application. With very few exceptions, components intended for use in mash-ups provide data but not a user interface. As a result, developers must typically create a UI from scratch. (The notable exception to this rule is the Google Maps API, which may hint at why 47% of the 2928 mash-ups listed on programmableweb.com put it to use.) Similarly, tools for creating mash-ups, such as Yahoo's Pipes, are typically data-centric: they facilitate remixing of data, but do not specifically address the question of tailoring the user interface of interactive web applications. Rather, Pipes and data-flow approaches in general assume that data itself is paramount, rather than the combination of data and a well-designed interface.

Related Work

Prior work in web tailoring falls into two areas: browser extensions for client-side tailoring and programmable proxies that interpose between the web browser and web servers.

Browser-side tools [1, 4] have the advantage of easily supporting the logged-in and AJAX-enabled web.

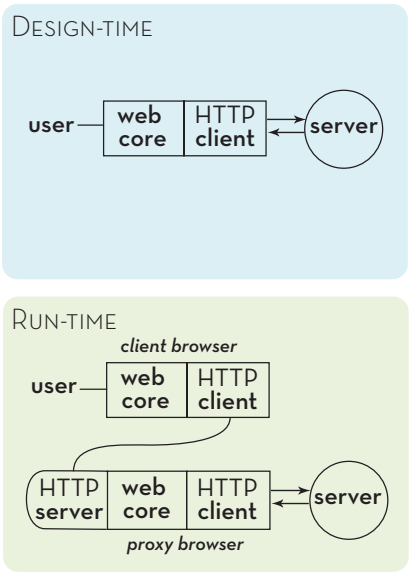

Figure 2. The re:mix architecture at design time and runtime.

Additionally, being browser-based facilitates direct manipulation editing. 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 are generally shared manually through an upload or synchronization process. Even when the sharing is automatic—as with Koala [4]—the customized web applications can only be accessed from browsers that have the customization tool installed. This points to an important shortcoming of current browser-based tools: tailored applications do not retain their original software-as-a-service nature.

In contrast, programmable web proxies [2] support tailored applications that retain their software-as-a-service architecture. Traditionally, however, these tools 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.

Tools are beginning to emerge which span this gap—Highlight is one such example [5]. In this system, web pages are tailored client-side using a browser plugin. These tailorings are then uploaded to a server which runs a special web browser that offers these tailorings as services. The major distinction between our work and Highlight is that Highlight supports *one* type of client-side tailoring. Our architecture is indifferent to the method of tailoring—users can use any Firefox extension they like to tailor web pages, and these pages are then served as a service by their computer to any other device.

Architecture Support for Tailoring

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 2). We have implemented the re:mix proxy inside POW, a Firefox extension that runs an HTTP server inside the browser.

At design time, users create remixes using the Firefox extensions 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. 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 automated deployment of tailored applications as a service. Currently, deploying these remixed applications is currently time consuming, and requires a great deal of technical web expertise. This is not surprising—our work thus far has focused on making third-party application tailoring possible, not on making it easier.

On a technical level, more work also needs to be done in securing re:mix, disclosing the nature of a re:mix script to a user, and in understanding the security implications of tailoring extant web applications. As it is, managing trust across web services is not easily done, but security researchers have recently proposed mash-up-level security mechanisms which may prove useful in future work.

In addition to addressing the needs of the mobile web, 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.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] Nichols, J, Z. Hua, J. Barton. Highlight: A System for Creating and Deploying Mobile Web Applications. In Proceedings of UIST: ACM Symposium on User Interface Software and Technology. ACM Press, 2008.