A Stateful Web Augmentation Toolkit

Matthew Webber, Robert C. Miller
MIT CSAIL
Cambridge, MA 02139
{wmatthew, rcm}@mit.edu

ABSTRACT

We present a toolkit that lets end-users extend content on database-backed web sites with live, personal data. The toolkit provides a modular, extensible framework characterizing the layout of content on a site, augmenting the data that is displayed, and storing the augmentation locally or in the cloud. This expands the range of possibilities for web end-user programming.

ACM Classification: H.3.5 [Online Information Services]: Web-based services

General terms: Design, Human Factors

Keywords: Mashups, End-User Programming

INTRODUCTION

Information on the web is highly structured. Much of it lies in databases; when a user visits a site, the database is queried and data is filled into a template and displayed. The structure helps content providers easily reconfigure their site presentation: elements can be reordered, augmented or filtered out with minor changes to query statements.

When technically savvy end-users want to reconfigure content presentation, they use mashups. Mashups are software applications that merge separate APIs and data sources into an integrated interface[1]. By making a mashup with a tool like Mashmaker[2] or Popfly[3], users can change the layout of content or filter out certain records. Multiple data sources can be mixed together; for example, one site's map layout can be used to plot another site's content. Users can do all this without assistance from content providers.

One thing that end-users cannot easily do is augment the underlying database. Mashups are generally stateless: they do not allow user-generated data to be tied with records, edited in place, and stored. They are limited to representing existing data and functionality, not expanding it.

On the backend, where data is structured into tables and accessed with queries, changing a site's functionality is possible. But end-users can't do what database administrators can, because back ends of sites are hidden from them.

Our work aims to empower end-users to extend the information presented on database backed web sites, in-place, with live, personal data. We present a toolkit that reproduces the functionality of modifying underlying databases that power a site. By augmenting records with arbitrary data, users can effectively add columns to the underlying table.

UIST'09, October 4–7, 2009, Victoria, BC, Canada.

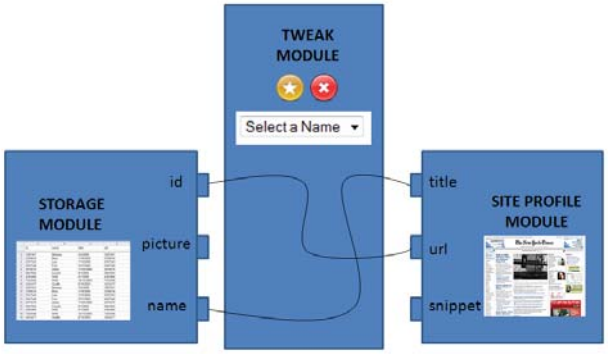

Figure 1: The Tweak Module maps selected fields from the Storage Module to fields in the Site Profile Module. The Tweak Module also adds UI elements so data fields can be changed inline.

By creating new records, they can effectively add rows. By filtering and combining data sources together, they can effectively rewrite the queries and joins that gather data.

As with mashups, these actions are fully controlled by the end-user; they can be initiated, modified and distributed without the participation, blessing or knowledge of content providers. This expands the range of possible activities for end-users, from changing layout to substantially changing site functionality.

IMPLEMENTATION

The Stateful Web Augmentation Toolkit (SWAT) is built on the Chickenfoot web scripting environment[4]. It has three modules: a Storage Module for storing user-generated and user-controlled content, a Site Profile Module for tracking the structure of a site's content, and a Tweak Module, that maps fields between the other two modules and modifies web pages.

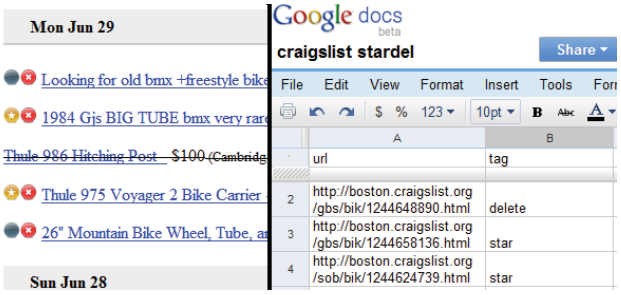

Figure 2: Left: An augmented craigslist page with two ads starred and one deleted. Right: the application's data stored in a Google Spreadsheet.

The **Storage Module** serves as a storage mechanism for new content, as well as an interface for mashing up content stored locally (e.g. a contact list or spreadsheet stored on disk). Most of the data the end-user interacts with is stored by content providers. The storage module only keeps track of user changes: the difference between the existing data and the modified state. This lowers storage costs for some applications and makes others possible where it would not be feasible to traverse data sets in their entirety. Since data is fetched live every time a page is accessed, the information displayed to the user remains as fresh as if that user was interacting with the site normally.

The **Site Profile Module** keeps track of the structure of a site: what URL patterns are relevant, which DOM elements contain information about records, and how to extract those records. This module needs to uniquely identify records. For many sites, a URL can provide a unique identifier: search results, product listings, and news pages often link to record-specific pages with an unchanging URL. For other sites, like AJAX applications, URLs are often not enough.

Other strategies, like using RDF data, searching for primary key fields hidden in the DOM, or combining immutable fields to create a primary key, will work on some sites and not on others. By keeping this part of the system behind an interface, different solutions can be made for different sites.

Different applications may have different concepts about what constitutes a distinct record. When a craigslist ad is reposted or a news article is rewritten, some may consider these events to be changes to existing records. Others may consider them to be new record creation. Each application can use a site profile module that fits its needs.

The **Tweak Module** associates selected fields of a Storage Module and a Site Profile Module. This module is also responsible for using the other two to add and remove records, and augmenting the HTML associated with a record to allow the end-user to change that record.

APPLICATIONS

This section explores modules used by two SWAT applications we have built, and discusses other possible modules.

The first application adds a delete button and a "favorite" star to craigslist listings. The Site Profile Module identifies HTML elements that correspond to craigslist listings. The Tweak Module injects two buttons (with associated javascript) beside each listing. When a listing is starred or deleted, the Storage Module adds a record with the primary key (the listing's URL) and appropriate status.

The storage module in this example is a Google Spreadsheet with two columns. Because the spreadsheet has a GUI editor, users can view and edit their data as they would any other spreadsheet. Because the data is stored in the cloud, the spreadsheet could be shared by a number of users at different locations who are collaborating on a group search.

A second application annotates Amazon.com product listing with a birthday gift tracker. On each product page, a drop-down menu box is added that draws names from the user's read-only contact list (a flat file on the user's machine). When a contact name is selected, their birthday is automatically populated in the date field. All selections are stored in a CSV file on disk.

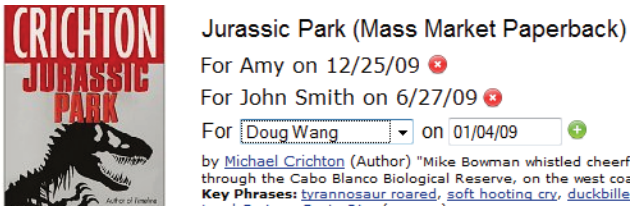

Figure 3: Amazon product listing augmented to include a birthday gift list. Names are drawn from a contacts file on disk.

Additional Storage Modules could be built to draw name and birthday data from other sources (e.g. Outlook, Facebook) and merge them together.

As described, these two applications do not share any modules, but their constituent modules could be recombined to make new applications. Amazon products could be starred. Gift lists could be stored in the cloud. Craigslist posts could be tagged with people's names. As more modules are implemented, the number of possible applications increases dramatically. When modules break, they can be replaced without rewriting entire applications.

FUTURE WORK

We wish to evaluate the technical limitations and usability of this toolkit. We plan to study the Site Profile Module's unique record identification performance across a number of kinds of web content- video, news, shopping, social networks, and search results.

It is also critical to evaluate how programmers who are not familiar with this project or the Chickenfoot web scripting environment fare at combining, extending and replacing existing modules to make applications with this toolkit. Ultimately, our goal of expanding the range of possibilities for web end-user programming relies on this toolkit being as transparent and easy to use as possible.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under award number IIS-0447800, and by Quanta Computer under the T-Party project. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

1. Zang, Rosson, and Nasser. Mashups: Who? What? Why? CHI 08.
2. Ennals and Garafalakis. Mashmaker: Mashups for the Masses. SIGMOD 07.
3. Microsoft Popfly. http://popfly.ms
4. Bolin, Webber, Rha, Wilson, and Miller. Automation and Customization of Rendered Web Pages. UIST 05.