Marmite: End-User Programming for the Web

Jeffrey Wong, Jason I. Hong
Human-Computer Interaction Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15217
+1 (412) 268-1251
jeffwong@cmu.edu, jasonh@cs.cmu.edu

ABSTRACT
The recent popularity of mashups has shown that people can recombine web services to create new and easier ways of accessing or aggregating information. However, creating mashups requires skills that are out of the reach of non-programmers. In this paper, we describe how the end-user programming approach might enable non-programmers to make mashups, discuss what obstacles must be overcome for this type of system to work, and present what we have learned from the our prototype so far.

1. INTRODUCTION
Much of the recent innovation behind mashups has been the recognition of useful ways of combining information and services from different websites to fulfill some unmet need or create simpler ways of carrying out tasks. A key problem here, however, is that **creating mashups requires a great deal of programming expertise** in areas such as web crawling, text parsing, pattern matching, and HTML. Thus, it takes a great deal of time and skill to create such services. Past work tends to emphasize low-level data processing or programming techniques that are beyond the ability of average web users.

Although the recent flowering of Web Services has made it easier for *programmers* to create mashups, there still is little support for typical end-users. Furthermore, programming may be uneconomical to solve some information problems that may be unique to a few individuals. Sharing these solutions and supporting them with server space and bandwidth may never happen if there is little expected return.

The goal of Marmite (see Figure 1) is to solve these problems by enabling users who have little or no programming experience to create personal mashups and providing a common platform to share these mashups. By making it easy to create and experiment with different combinations of web services and share them, mashup programmers can observe what needs are being unmet and create more refined services to meet these needs. More specifically, Marmite will let end-users:

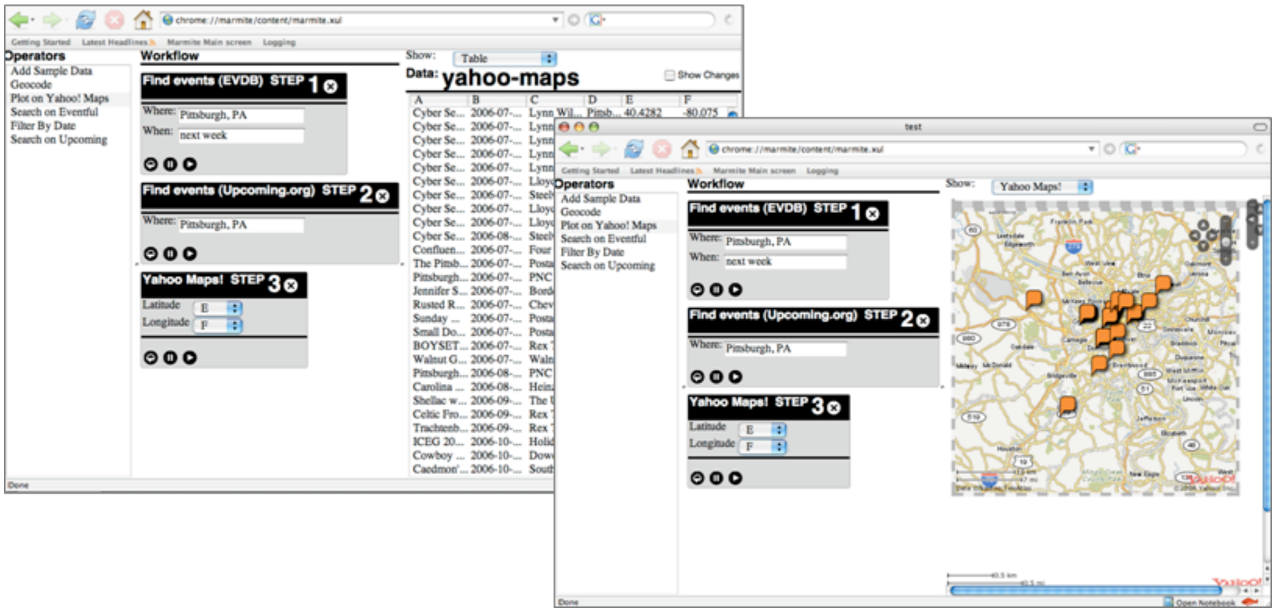
- Easily extract interesting content from one or more web pages (for example, names, addresses, dates, phone numbers, URLs, and other kinds of data types)
- Process it in a data-flow manner, for example filtering out values or adding metadata
- Integrate it with other data sources, either from local or remote databases, from other already existing web pages, or already existing services (similar to a database join operation)
- Direct the output to a variety of sinks, such as databases, map services, text files, web pages, or compliable source code that can be further customized

We are developing Marmite to support a dataflow architecture, where data is processed by a series of *operators* in a manner similar to Unix pipes. For example, one scenario Marmite will support is, "find all of the addresses on this set of web pages, keep only the ones in Pennsylvania, and then put them all onto Google Maps." Another scenario that would let end-users create custom calendars is, "Every week, crawl through these five different web sites, extract all calendar events, sort by date, and publish it on our intranet as a custom web page." We are also designing Marmite so that it can be integrated with Web Service Description Language (WSDL) files, making it easy to integrate Marmite data flows with **web services** that have publicly available WSDL descriptions such as Google search, Google calendar, Amazon book search, EBay, and FedEx.

Since the design space is quite large, we are focusing our energies on several key research issues:

- **Specifying what data to extract**. In our preliminary user studies, the greatest barrier for end-users has consistently been how to select what web pages and what content on those web pages they want. Currently, we are using simple and well-known algorithms for finding and specifying patterns in web pages, but it is not yet clear whether these will be simple enough or sufficient for end-users.
- Designing, implementing, and evaluating a **hybrid dataflow / spreadsheet model** that makes it easy to see what the current state of the extracted data is and provides before-after views to help debug errors. Our preliminary user tests suggest that people like and understand the basic model, but it is not yet clear how well it will work for large data sets.
- Handling **errors in parsing and dataflow processing**. The UI needs to make clear what errors there are and provide graceful ways of recovery, for example, ignoring the errors, trying an alternative operator, or having the end-user manually fix the error.

The challenge for each of these risks is to provide a usable UI coupled with effective algorithms. We are taking several approaches to manage these risks, including starting with simple approaches first, getting quick and constant feedback from prototypical end-users to ensure that our solutions work, and rapidly iterating on what we learn.



A	B	C	D	E	F
Cyber Se...	2006-07-...	Lynn Wil...	Pittsb...	40.4282	-80.075
Cyber Se...	2006-07-...	Lynn			
Cyber Se...	2006-07-...	Lynn			
Cyber Se...	2006-07-...	Lynn			
Cyber Se...	2006-07-...	Lynn			
Cyber Se...	2006-07-...	Lloyd			
Cyber Se...	2006-07-...	Steel			
Cyber Se...	2006-07-...	Lloyd			
Cyber Se...	2006-07-...	Lloyd			
Cyber Se...	2006-08-...	Steel			
Confluen...	2006-07-...	Four			
The Pittsb...	2006-07-...	Posta			
Pittsburgh...	2006-07-...	PNC			
Jennifer S...	2006-07-...	Bord			
Rusted R...	2006-07-...	Chev			
Sunday ...	2006-07-...	Posta			
Small Do...	2006-07-...	Posta			
BOYSET...	2006-07-...	Rex			
Walnut G...	2006-07-...	Waln			
Pittsburgh...	2006-08-...	PNC			
Carolina ...	2006-08-...	Heinz			
Shellac w...	2006-09-...	The			
Celtic Fro...	2006-09-...	Rex			
Trachtenb...	2006-09-...	Rex			
ICEG 20...	2006-10-...	Holid			
Cowboy ...	2006-10-...	Dow			
Caedmon'...	2006-10-...	South			

Figure 0. Two examples from the current implementation of Marmite. The left shows a small set of operators that can be chained together, a dataflow consisting of three steps, and a spreadsheet showing extracted values. The right shows these values being placed on top of a map.

2. EXISTING APPROACHES

Creating a data-flow to manipulate many pieces of data often requires a user to engage in some form of programming. Research on end-user programming has found that programming is difficult for novices for a number of reasons [11]: it is difficult to enter syntactically correct code; finding appropriate operations is difficult; and it is hard for novices to understand their own programming errors [12]. There are a variety of solutions to these problems (for a review, see [9]). Marmite minimizes code entry problems by having users work with graphical dialog boxes that represent operations. It also helps prevent errors by allowing users to incrementally add steps to their program and observe changes to their data.

Another problem in web automation is extracting data from a page. In our experience, one of the key challenges of end-user programming for the web is making it easy for typical end-users to specify what parts of a page or set of pages they want, while being flexible enough so that a variety of tasks can be supported.

Identifying patterns of relevant information on a web page can be done with web page parsing APIs, frameworks for existing programming languages [8], or specialized languages [2] [10] but require programming and HTML knowledge. Recent work has attempted to mitigate these problems. Chickenfoot [3] can match text using natural language expressions (e.g. “just before the text box”) but still requires programming in Javascript. C3W [6] is a point-and-click tool to feed data into web applications but doesn’t scale well beyond a handful of items, doesn’t make operations obvious, and cannot easily extract multiple pieces of data. PiggyBank [7] extracts data from websites that are augmented with semantic data but requires JavaScript to extract data from normal websites. We are designing Marmite to avoid these issues by having users interact with a set of pattern-matching algorithms that automatically locate lists based on regularities in HTML structure and content-based heuristics, thus requiring no programming. For our current implementation, we are using a simple pattern-matching approach similar to that used by PiggyBank [7].

There are some commercial tools that automate tasks or perform mass operations on text data. Anthracite [13] and Apple’s Automator [1] visually represent the automation but require knowledge of the HTML structure of web pages. However, feedback in these tools is poor since users must execute their programs in their entirety to verify that they are correct. It is also difficult to see intermediate results in both of these applications as well, as they tend to focus on the program rather than the data. Marmite avoids these problems through incremental program construction and manipulation of graphical objects representing code, as well as a spreadsheet view that shows the data as it is being extracted from web pages and modified by Marmite’s data flow.

More recent work in the mashup community includes Dapper [4] and DataMashups [5]. Dapper is tool for creating screen-scrapers that allow you to access any web page as if it was structured data in an XML document. Although it is an excellent way to avoid having to write a screen-scraper, this tool is geared more towards programmers who can make use of its output. DataMashups is a visual programming environment that lets one create a website with common widgets that one might find in a web application. However, we believe that this is similar to tools like Visual Basic and over-emphasizes the construction of interfaces rather than data manipulation.

3. CURRENT STATUS

We have created and evaluated 6 different mockups of Marmite, and are currently developing our first interactive prototype of Marmite as a Firefox extension using Javascript and XUL, a user interface description language for the Firefox web browser.

We are also developing operators as WSDL-based web services, meaning that operators can be developed in any programming language.

We are currently in the process of running user tests with our interactive prototype. Here, we examine the three research challenges we are trying to address with Marmite, and informally describe what we have learned thus far.

With respect to specifying what data to extract, we believe this is a general problem that all mashup tools will face. The reason this is difficult is because there is such a wide range of possibilities. For example, by selecting a single link from a page, a person might want just that link, all links in that group, all links on that page that are not part of the overall navigation, all links on that page, or all of the aforementioned possibilities but for a set of pages. It is possible to apply well-known machine learning techniques or programming by demonstration techniques here; however, the challenge is to find the right combination of simplicity and flexibility that will help end-users succeed.

With respect to the hybrid dataflow / spreadsheet model, the observation here is that most programming environments are either code-centric, focusing on the program itself (for example, Java, Visual Basic, and Automator [1]), or on the data (for example, spreadsheets). Our goal with the hybrid dataflow / spreadsheet model was to develop an environment that supported both, providing the flexibility of programming while also providing concrete examples of data rather than abstract notions that may be difficult for novices.

With respect to handling errors in parsing and dataflow processing, we have not yet addressed errors in parsing, but have encountered problems with end-users and dataflow processing. Basically, the errors are either uncertainty as to how to start in a new dataflow (a variation of the input problem), as well as what to do next. For the latter issue, we have been adding simple data type inferencing to Marmite, for example recognizing things like street addresses and phone numbers, so that Marmite can also suggest what kinds of operations an end-user might want to apply next. This makes it easier for people to go through the list of existing operators, as only the relevant ones are shown.

4. REFERENCES

[1] Apple Automator. http://www.apple.com/downloads/macosx/automator/

[2] Barrett, R., Maglio, P., and Kellem, D.. “How to Personalize the Web.” *Proc. CHI’97*, pp. 75–82.

[3] Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R. C. Automation and customization of rendered web pages. *Proc. UIST '05*. ACM Press (2005), 163-172

[4] Dapper. http://www.dappit.com

[5] DataMashups. http://www.datamashups.com

[6] Fujima, J., Lunzer, A., Hornbæk, K., and Tanaka, Y. Clip, connect, clone: combining application elements to build custom interfaces for information access. *Proc. UIST '04.*, ACM Press (2004), 175-184.

[7] David Huynh, Stefano Mazzocchi, and David Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. International Semantic Web Conference (ISWC), November 2005, Galway, Ireland.

[8] Miller, R. and Bharat, K. SPHINX: A Framework for Creating Personal, Site-Specific Web Crawlers. *Proc. WWW7*, (1998), 119-130.

[9] Kelleher, C. and Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv. 37*, 2 (2005), 83-137.

[10] Kistler, T. and Marais, H. WebL - a programming language for the Web. *Proc. WWW7*, (1998) 259-270.

[11] Ko, A. J., Myers, B. A., and Aung, H. Six Learning Barriers in End-User Programming Systems. *IEEE Symp. On VLHCC*, (2005) 199-206.

[12] Ko, A. J. Myers, B. A. Human Factors Affecting Dependability in End-User Programming. 1st Workshop on End-User Software Engineering (2005), St. Louis, MI, 1-4.

[13] Metafy Anthracite. http://www.metafy.com/products/anthracite/