Marmite: End-User Programming for the Web

Jeffrey Wong
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15217 USA
jeffwong@cmu.edu

Jason Hong
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15217 USA
jasonh@cs.cmu.edu

Abstract
A tremendous amount of semi-structured data is available today on the web but is not necessarily in a form which is suitable for a user's tasks. For example, a website may show a listing of local events but a user wants to filter out those which are too far from him. To address this problem, we are developing a tool called Marmite that helps users extract data from web pages and create new applications using a dataflow architecture in a manner similar to Unix pipes. In this paper, we describe formative user studies, some evaluations of low-fidelity prototypes, and a set of design recommendations for this tool.

Keywords
End-user programming, web automation, web browsers

CHI 2006, April 22–27, 2006, Montreal, Canada.
ACM 1-xxxxxxxxxxxxxxxxxx.

ACM Classification Keywords
H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces; D.2.6 [Programming Environments]: Interactive environments;

Introduction
Imagine a visitor to a new city has found a web page listing recommended restaurants. He wants to get a sense of where they are, but the web site does not provide an easy way of doing this. With a standard web browser, he would have to copy each address, paste it into a separate map service, and return to the list to get the next address. This task could be done more efficiently if the user could tell his computer how to extract relevant information from the web page and enter it into the mapping application for him. In short, we wish to address this problem with a tool which enables *end-user programming on the web*.

Because web developers cannot foresee all possible needs of end users, users must find cumbersome workarounds to get the information they want. Web services APIs (WSAPIs) partially alleviate this problem by making the content and functionality of websites accessible, but require systems-level programming, which is difficult for average users.

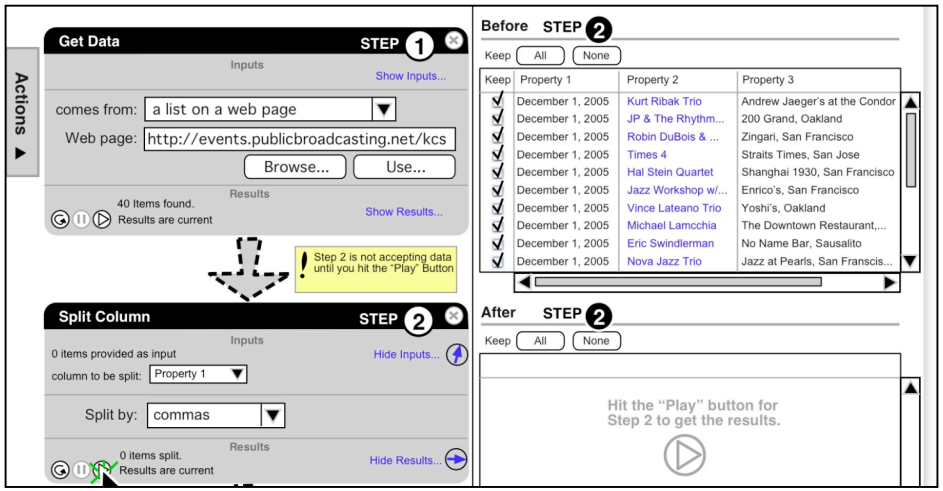

Figure 1. This figure shows the conceptual design for Marmite. The left side of the screen shows operators that have been added to the dataflow. The right side is a space where the before-after views of the data can be shown for each operator.

In this paper, we present a design for a tool called Marmite that addresses two major problems in this domain: 1) extraction of relevant data from the web and 2) helping the user build a program which manipulates the data correctly to complete his task. Extracting relevant data without being very explicit (i.e. cutting and pasting) is difficult, repetitive, and tedious. Data also has internal structure, which needs to be identified in order for a computer to work with the data (i.e. an event has a name, a place, and a time). Our design addresses this problem by looking for patterns on a web page using a combination of typographical and content-based heuristics and asking the user to choose the pattern that matches their intention.

Our design represents programs both as data-flows and as a spreadsheet, letting users observe their data as it changes (see Figure 1). Programs are graphically represented as a series of small steps, or "operators", which transform the data. Examples might include "selecting only messages from a particular person" or "compute the time left until each event." Marmite supports iterative, interactive programming by providing before and after views (similar to those in the StageCast system [14]) which show the effects of operators on live data.

This paper describes the user studies and design rationale that led to Marmite's design and outlines design recommendations which we will use to for our first hi-fi prototype.

Related Work

Our work applies research on end-user programming to improve upon existing web automation and commercial end-user programming tools.

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 [10]: 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 [11]. There are a variety of solutions to these problems (for a review, see [8]). 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 lists of data from a page. Most web pages do not provide semantic tags to find, for instance, all phone numbers on a web page. Humans can recognize lists of

information based on semantic and/or typographic patterns. Identifying patterns of relevant information on a web page can be done with web page parsing APIs, frameworks for existing programming languages [7], or specialized languages [2] [9]. These approaches, however, 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 [5] is a point-and-click tool to identify data and input it into other web applications. However, C3W doesn't scale well beyond a handful of items, doesn't make operations obvious, and cannot easily extract multiple pieces of data. 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 experience.

Finally, there are some commercial tools which automate tasks or perform mass operations on text data. iOpus Internet Macros [6] records user behaviors but requires the user to edit code to create patterns. Anthracite [12] visually represents the automation but requires knowledge of the HTML structure of web pages. Apple's Automator end-user programming environment [1] suffers from similar problems. Feedback in these tools is poor since users must execute their programs in their entirety to verify that they are correct. Marmite avoids these problems through incremental program construction and manipulation of graphical objects representing code.

User Studies

Our design process consisted of a user test of Apple's Automator [1], a blank paper study, and low-fidelity paper prototypes. The tasks used in our studies had a similar structure: extract some data from a web page, perform some kind of processing on the data, and output the results. Because knew we wanted to create something like Apple's Automator (but more focused towards data extraction from the web), we conducted a user test to identify its usability problems for our task. To design a method for interactively extracting text from a web page, we conducted a blank paper study, inspired by similar studies in natural programming designs [13], where we asked users to write down instructions that they believed would unambiguously extract text. We then iterated through six paper prototypes with twenty users.

AUTOMATOR STUDY

As our initial prototypes began to resemble Apple's Automator tool, we first conducted a user test with Automator to anticipate the kinds of usability problems users might encounter. We created 3 tasks. The first task was a simple warmup task and the other two were that were mass operations that involved traversing links and downloading images. We found the following problems:

- Users had no feedback about what the state of the data was in-between steps.
- For a mass operation, once a program that visited many web pages had been constructed, executing it is time consuming. Detecting problems early in the data-flow makes problems easier to understand.

- Users generated theories about why problems occurred but were not very good at coming up with theories to test them. This is consistent with prior work on EUP [11].
- Operator organization and grouping was a factor which affected whether the user could find the right operator.

Blank Paper Study

Our goal for the blank paper study was to understand users' concepts of text patterns and provide inspiration for natural interaction methods for selecting text patterns. Three participants were asked to write unambiguous instructions for another person that would extract multiple instances of a data type (such as a company or hotel name). They used semantic references such as "get all of the names of companies." Some participants used drawings and referred to the typographical features (e.g. "all of link text up until the hyphen"). Based on these observations, we decided it would be simpler to automatically detect text which appears to be in a list and let the user choose amongst some guesses before delving into more manual methods of specifying the text. Recent work in HTML pattern detection [4] suggests that this is possible.

Lo-fi Prototypes

We conducted six rounds of paper prototypes with twenty participants. The primary task for our design was to extract a list of restaurants and addresses from a local nightlife website, get geographic coordinates for those restaurants, and plot those locations on a map. Figure 1 shows an example of one of our paper prototypes.

Final Design Strategies

Based on these prototypes, our studies, and prior work, we arrived at some design strategies which we will be employing in our implementation.

Hybrid Data-Flow/Spreadsheet View

Our design combines the advantages of data-flows and spreadsheet views of the data. Data-flow views, as seen in Automator [1] and Anthracite [12], allow users to get an overview and add new operations to quickly build new programs. Spreadsheets, on the other hand, can show the effects of operations on a data set immediately. Our design combines a data-flow view of the program with before-after views [14] of the data, which can help minimize errors. Data items are organized rows of a table and each column is an semantic attribute of the data item (e.g. name, address, or phone number). Operators change the state of this table.

Encapsulate Interactions with the Web in Operators

A goal of our project is to create data-flows that connect to web services that provide useful data processing services, such as mapping, product code lookup, or currency conversion. In our design, these connections are encapsulated in operators that abstract away the complexity of providing input to web apps and extracting their results.

We made the simplifying decision that operators which access web services could be created by more skilled users using tools like Chickenfoot [3] or WSAPIs and then shared with other users of our tool. Later, we may provide an interface which allows average users to create their own operators.

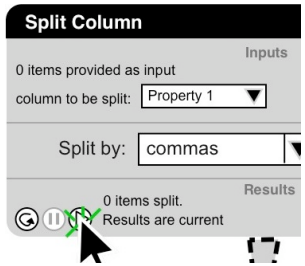

Figure 2. Incremental execution at the level of each operator is supported by the Resume, Pause, and Play buttons.

Interactively Detect Relevant Data

It is difficult to devise an intuitive interaction which unambiguously lets a user select all data fitting a pattern but does not require the user to be completely explicit about the data required or create a pattern syntax. We decided that Marmite find pattern in the HTML structure of the page and the content and then ask the user select the desired pattern (see Figure 3). Although users had no difficulty with this type of style of selection, we would have to verify whether the algorithms [4] in practice could select relevant content across a variety of information presentation styles.

Support Incremental Execution

Marmite lets users control when their operators will work on the input because some operators need to do time-consuming things like visit another website for each data item.

When users add a new step, the operator is not executed immediately on the input data because settings for the operator may need to be adjusted.

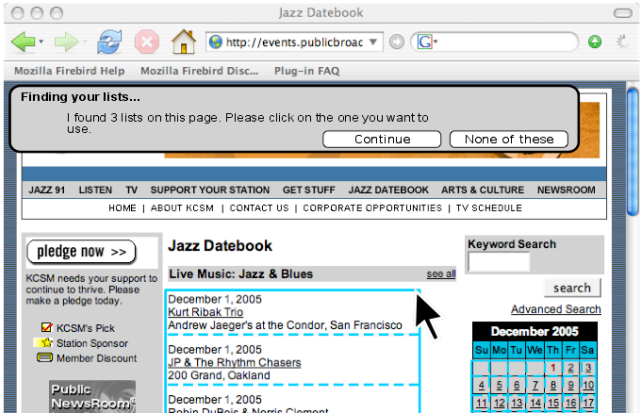

Figure 3. Marmite will allow users to select relevant list items in a web browser.

Each operator comes with Reload, Pause, and Play buttons (see Figure 2). The Pause button halts the execution of the operator. This supports testing the entire data-flow quickly because the user can halt the execution of an operator after it has finished executing on a small sample of the inputs. The Reload button lets the user re-run the operator on the input provided to it.

The Play button enables the operator such that any input passed to it is automatically acted upon and placed in the output for the subsequent operator. Marmite also provides depth-first previews. If there is a set of data that needs to be passed through a series of steps, Marmite will pass each item through as many steps as possible before working on the next item.

Suggested Next Actions

Marmite offers suggestions for actions based on the guesses of the data types of the results of the last operation because, in our Automator study, we found that users had trouble locating and selecting a next action.

Support Reuse

Data-flows constructed by Marmite users represent new functionality if the data-flows can be packaged and made available to others. Ideally, these packaged data-flows should be able to take parameters so they can be customized to other users' needs. For example, a data-flow that takes a list of locations and computes the distance from the user's current location might have an operator that lets the user enter an origin point in the Marmite interface. However, from our lo-fi prototypes, we found that the idea of providing parameters to a data-flow is a difficult concept for novice users to

understand. We have yet to verify that the affordances we provide in our most recent prototype make sense.

PROVIDE TEMPLATES FOR COMMON TASKS

Marmite will provide templates to solve common tasks such as plotting a set of locations from a web page onto a map. Templates will help users overcome the barrier of design and action selection. Also, looking at templates is similar to looking at sample code in other programming environments, which has been found to be common strategy for novice users when learning a new programming environment [10].

Implementation

Our next step is to prototype a pattern extraction system to verify that the pattern identification interaction works for most web pages. We will then build an interactive prototype to test our overall concept using hand-built operators. Then build a framework for constructing operators that work with any web application, and make that framework public. We hope that the new and interesting operators will be created by users trying to solve their own problems.

Conclusion

We have designed an end-user programming environment for data-flow processing which is we believe is worthy of further validation and user testing. By making providing a way for end-users to create new data-flows, we hope that Marmite will help ordinary computer users create new programs to meet their needs.

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] Chang, C. and Lui, S. IEPAD: information extraction based on pattern discovery. *Proc. WWW10*. ACM Press (2001), New York, NY, 681-688.

[5] 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.

[6] iOpus Internet Macros. http://www.iopus.com/

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

[8] 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.

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

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

[11] 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.

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

[13] Myers, B. A., Pane, J. F. and Ko, A. Natural Programming Languages and Environments. *Comm. of the ACM*, (Sept. 2004), 47-52.

[14] Smith, D. C., Cypher, A., and Tesler, L. Programming by example: novice programming comes of age. *Comm. of the ACM* (Mar. 2000), 75-81.