Making Mashups with Marmite: Towards End-User Programming for the Web

Jeffrey Wong, Jason I. Hong
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
jeffwong@cmu.edu, jasonh@cs.cmu.edu

ABSTRACT
There is a tremendous amount of web content available today, but it is not always in a form that supports end-users' needs. In many cases, all of the data and services needed to accomplish a goal already exist, but are not in a form amenable to an end-user. To address this problem, we have developed an end-user programming tool called Marmite, which lets end-users create so-called mashups that re-purpose and combine existing web content and services. In this paper, we present the design, implementation, and evaluation of Marmite. An informal user study found that programmers and some spreadsheet users had little difficulty using the system.

Author Keywords
mashup, end-user programming, web, spreadsheet, user study

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
There is a tremendous amount of web content available today, but it is not always in a form that supports end-users' needs. For example, it is easy to find a list of hotels in San Jose, but it is not so easy to sort them by distance to the San Jose convention center. To do this today, an end-user would have to manually enter in each address into a mapping service and write down the distances for each, or manually construct software to do the same. There are two key observations here. First, all of the data and services needed to accomplish the goal above already exist, but they are not in a form amenable to her needs. Second, it is extremely unlikely that a web site will be able to support all the needs of all of its end-users. For these cases, we argue that it is better to provide tools that can help end-users help themselves.

What is interesting is that there is a rapidly growing community of web programmers creating so-called "mashups" that combine existing web-based content and services to create new applications. One of the earliest and best known mashups is housingmaps.com, which crawls rental listings from the Craigslist community web site and puts them on top of Google Maps. Rather than having to go through each of the text listings and manually entering each address into a map service, this mashup makes it easy for end-users to see a map of all the available rentals.

The desire to create mashups is strong. As of September 2006, ProgrammableWeb.com listed 1019 mashups created since the housingmaps.com mashup was deployed in April 2005, which is fairly rapid growth by any metric.

A key problem here, however, is that creating mashups

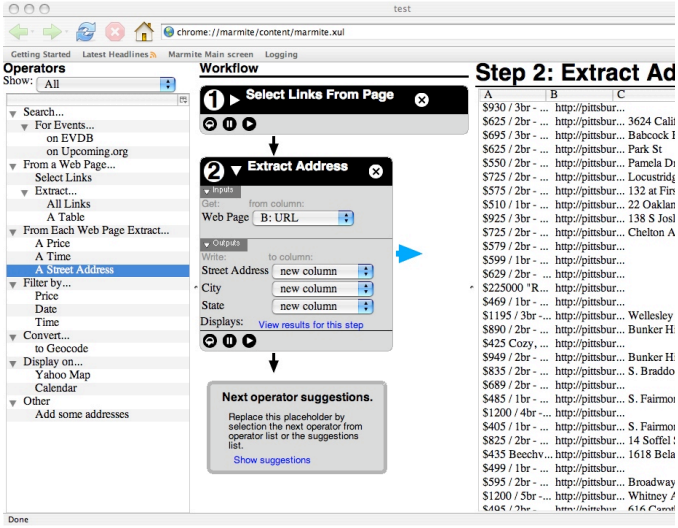

Figure 1. Marmite is an end-user programming tool for creating web-based mashups. On the left is a set of operators that users can select to extract and process data from web pages. In the middle is the data flow, and on the right is a spreadsheet view that shows the current values of the data.

requires a great deal of programming expertise in areas such as web crawling, text parsing, pattern matching, databases, and HTML. Thus, it takes a great deal of time and skill to create such services. Most tools to help end-users create mashups tend to emphasize low-level data processing or programming techniques that are beyond the ability of average web users. Furthermore, most end-user programming tools tend to focus on GUIs (e.g. automating repetitive actions) rather than processing existing web-based content and services, and are thus inappropriate for these kinds of online tasks.

To address this problem, we are designing, implementing, and evaluating Marmite, a tool that will let everyday end-users create these kinds of mashups (see Figure 1). Marmite supports a data flow architecture, where data is processed by a series of *operators* in a manner similar to Unix pipes. For example, one scenario we are designing Marmite to 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 them by date, and publish them on our intranet as a custom web page."

More specifically, Marmite lets 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 existing web pages or 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 compilable source code that can be further customized

In this paper, we present the design, implementation, and evaluation of Marmite. We introduce a linked data flow / spreadsheet view that shows people how the data is being processed and what the current values of the data are. In other words, Marmite's linked view shows both the program and the data simultaneously. Our initial user study showed that some users were able to use this system and to construct programs with web services quickly without programming.

RELATED WORK

In this section, we position Marmite with respect to previous work in end-user programming, selecting content from web pages, website integration, the semantic web, and web services.

End-User Programming

There has been a great deal of past work in end-user programming. For example, Cypher [5] and Lieberman [20] describe numerous programming-by-demonstration and programming-by-example systems. Kelleher and Pausch [13] have also published a survey of programming languages and environments for novice programmers.

Creating a data flow to manipulate large data sets 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 [16]. They must: enter syntactically correct code, find appropriate operations, and understand any programming errors that may arise[15]. There are a variety of solutions to these problems (for a review, see [13]). Marmite minimizes code entry problems by having users work with graphical dialog boxes that represent operations. While this does limit what users can do, it helps prevent errors by allowing users to incrementally add steps to their program and observe changes made to their data. Marmite also suggests what operators they might want to use next, based on what data is currently being processed.

The closest existing work to Marmite in the area of end-user programming is Apple's Automator [1], a tool that lets end-users create scripts for automating repetitive actions. Automator provides a large set of reusable operations that can be assembled into a data flow. Marmite was inspired by Automator, and builds on its ideas by focusing on end-user programming for the web rather than the desktop GUI, re-purposing existing content and services rather than just repeating actions, and providing operations for parsing content. Marmite also introduces a linked spreadsheet/data-flow metaphor to help end-users understand what the current state of the data is.

An example of a programming system from the mashup community is DataMashups [7]. DataMashups is a visual programming environment that lets the user create a website with common widgets that he or she might find in a web application. However, DataMashups is more similar to interface builders like Visual Basic than a data extraction and processing tool.

Website Integration

Anthracite [22] is a "web mining desktop toolkit" with design goals similar to our own. It provides a graphical programming environment that lets users create data-flow automation over web pages. However, Anthracite is inaccessible to many end-users because using it requires understanding systems concepts such as regular expressions, databases, and HTML. While Anthracite is an improvement over the textual programming tools, our goal with Marmite is to lower the bar even more by simplifying the programming model, providing suggestions as to what the next step is, and providing better intermediate feedback so that end-users do not have to execute programs in their entirety to verify that they are correct.

C3W provides an interface for automating data flows in web applications [10] using the metaphor of spreadsheet

cells. With C3W, users can include functionality from any web application where the user fills out a web form and receives a result page. User can encapsulate a web form into a function by creating "cells" that map to the inputs of the form. The user then tells C3W how to find the data of interest in the resulting page and assigns a function name to encapsulated form. The function can then be used in spreadsheet formulas. For example, the user can take a company's stock trading symbol as input, find its dollar price by creating a function out of a web page that looks up the stock quote, and then convert this price into Japanese Yen by flowing this data through another web site.

Hunter Gatherer [31] and the Internet Scrapbook [33] are tools that enable users to interactively extract components of multiple web pages and assemble them into a single web page or a collection.

Our work differs from these systems in that we focus on extracting masses of typed data from web pages or even entire web sites, identifying data types of attributes of each item of data, providing a set of operations for the relevant data types found, and applying operations to a large set of data. For example, a listing of movies from a web site may be spread over several pages and have attributes such as the title (a string), rating (a number), and running time (also a number). When data is extracted from web pages and is broken down into constituent attributes and types, users can manipulate extracted web content with more flexibility and power. Listings can be sorted or filtered by an attribute and attributes can be operated on. If we have a listing of books with title and publication date attributes, we can look up price for the correct edition of each book in a bookseller's web service.

Hunter Gatherer and Internet Scrapbook focus on extraction of textual clips from web pages, without specifically breaking down extracted web pages into data items with typed attributes. Although C3W can make a function out of nearly any web form, it does not appear to be easy to work with larger sets of data (for instance, applying a function to a large listing of items). Users must invoke functions using a syntax for spreadsheet formulas. Furthermore, our Marmite focuses on providing an interface to web services. Some web services provide access to the core functionality that may be more flexible and lower-level than what a user would encounter using the web interface. Working with and remixing the sub-components of multiple websites is how most programmers create mashups.

Web Content Selection

We argue that one of the key challenges of end-user programming for the web is making it easy for end-users to specify what parts of a page or set of pages they want. This is difficult 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.

Identifying patterns of relevant information on a web page can be done with web page parsing APIs, frameworks for existing programming languages [25], or specialized languages [2, 14]. There has also been a great deal of work in developing *data wrappers,* programs that extract data from a web site so that it can be manipulated by traditional database systems [18, 19]. However, most of these tools are aimed at programmers.

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 requires programming in JavaScript. PiggyBank [11] extracts data from websites that are augmented with semantic data but requires JavaScript to extract data from normal websites. There has also been a great deal of work in automatically inferring structure in freeform text (for example, [29, 30, 23, 24, 4, 8]). Our goal with Marmite is not to innovate in these areas, but rather to use these algorithms to facilitate data extraction. Our current implementation of Marmite uses a pattern-matching algorithms from Sifter [12] and Solvent [32] , which uses XPath [34] queries to help the user select and extract multiple data of interest from a single page. Creo and Miro [9] compose a system which helps users extract information using a combination of programming-by-demonstration (PBD) and semantic libraries [21].

Dapper [6] is a tool for creating screen-scrapers that allows a user 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, it is geared more towards programmers who can make use of its output.

The Semantic Web and Web Services

There is also some related work in the area of the Semantic Web [34], an effort to embed machine-readable meaning and semantics to the large body of information that exists on the web. The goal is to transform this knowledge into a machine readable form so that computers can reason based on this information. Our goal with Marmite is orthogonal to Semantic Web efforts. If the Semantic Web takes off, then we can later adapt Marmite so that it can make use of those technologies. However, it is important to point out that there is a great deal of existing content already available, and that even if the Semantic Web takes off, we still have the same problem as before, namely that content and service providers cannot always predict all end-users' needs in advance.

Web services are a programmatic interface to the web, using the web as a medium for providing services through well-defined APIs with clear semantics (meant for

computers) rather than just HTML (meant for people). The problem is that web services are again aimed at programmers rather than end-users.

FORMATIVE DESIGN

To quickly explore the design space, we conducted a series of small studies to inform our design, consisting of a user test of Apple's Automator [1], a blank paper study, and a series of low-fidelity paper prototypes.

Automator Usability Study

Marmite was inspired by Automator, but is focused more on extraction and processing of web data. Thus, we felt that a running basic usability test with Automator would help us understand some potential usability problems that Marmite might encounter.

In Automator, end-users create a workflow by chaining together a series of operations. End-users can choose operations from desktop applications they are familiar with, such as getting email addresses from the address book application or retrieving a web page with the the Web browser. We asked three participants to complete three tasks. The first task was a simple warm-up task and the other two were operations that involved traversing a large number of links and downloading a large number of images.

We found the following problems:

1. Participants had a hard time knowing what operation to select. This was especially when for creating a new dataflow. Although we showed our participants several examples of how to chain operations together, they often had a hard time selecting the first operation to get started.

   Solution: Suggest relevant next actions. In our prototypes, we developed a few ways of having Marmite suggest what kinds of operations they may want to do next, based on the data currently available. For example, if there was street address data available, Marmite would suggest generating a map as one of the next operations.

2. Participants had no feedback about what the state of the data was in between operations.

   Solution: Show results of intermediate steps. Many end-user programming tools take either a program-centric view, where the program itself is the main view, or a data-centric view (most notably spreadsheets), which emphasizes the data over the program. In our paper prototypes, we developed a linked data flow / spreadsheet view that shows the program itself (data flow) as well as the effects of a given step in that data flow (spreadsheet view).

3. Since it can take a large amount of time to execute programs that involve large amounts of data, it was difficult to rapidly iterate on programs and ensure that the programs do the right thing. This was especially problematic for programs that copied or downloaded files, since the copies would have to be deleted before trying again.

   Solution: Support incremental execution. Participants in the Automator study and in our paper-prototype study found the operators fairly easy to understand and use, as they made programming easier by letting people build programs incrementally and correct errors earlier.

4. Participants often generated theories about why problems occurred but were not effective at coming up with theories to test them. This is consistent with prior work on end-user programming [16].

   Solution: Incremental execution and showing intermediate results helps alleviate this problem somewhat because it is easier to see where the problem occurred. Automatic generation of theories for programming bugs is currently an open problem in software engineering [17].

Blank Paper Study

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 [28]. In these blank paper studies, three participants were asked to write down unambiguous instructions for another person that would extract multiple instances of a data type (such as a company or hotel name) from a web page. 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").

This finding suggests that since users would most likely extract information based on semantics, typographic, or other features that can be difficult for a computer to understand or even ambiguous for a human being, the system must use a variety of heuristics, data detectors, or semantic detectors [9] to come up with some guesses about what data the user wants. The system and the user can then engage in a negotiation to navigate between guesses and the parameters of guess to arrive at what is actually desired (or close to it).

> We address this problem in Marmite's design by having the user interactively detect relevant data. The system should try to guess what patterns of information are desired and find the user's intent is through negotiation. This feature is only partially supported in the current implementation of Marmite.

Paper Prototypes

We conducted six rounds of paper prototypes with twenty participants. The paper prototype allowed users to assemble a sequence of operators (a data flow) that performed some tasks on a set of data. When an operator executes, its output

is available as input for the following operation. Users could see the results of operations in tables to the right of the data flow. Each operator was associated with a before-and-after view of the data.

Our vision for our design was that users would be able to interact with web services as graphical operators. The operators would be built and connected to web services by programmers who could add their operators to Marmite operator pool. Some of these might be provided by web service providers themselves, who might be interested in increasing the use of their services.

As we were evaluating the paper prototypes, we noticed that even though users might understand how Marmite worked conceptually, the usability of Marmite was still very sensitive to the design of the individual operators. This was a major concern Marmite's operators would be created by programmers. We decided to address this in our high-fidelity prototype by creating a code framework where the interface was generated for the developers simply by specifying their inputs and outputs. This enables Marmite operators to be consistent and lets programmers concentrate on connecting to the web services.

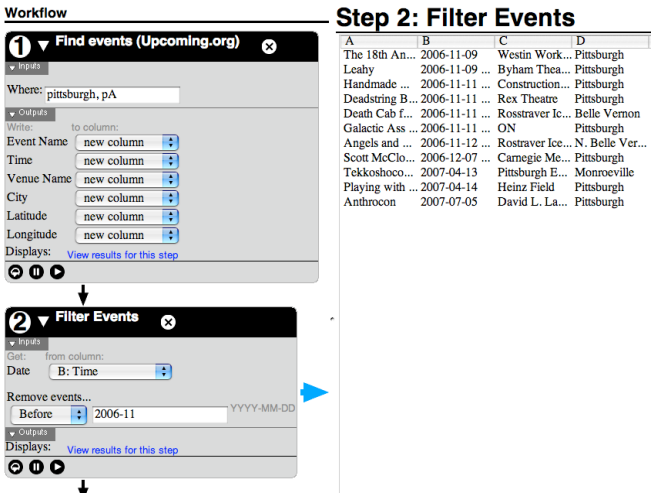

Figure 3. A data flow showing an operator which retrieves event records from a web service and passes it to an operator which filters the records by date. Note that this screenshot does not show the operator selection area, that is the set of possible operators that can be added.

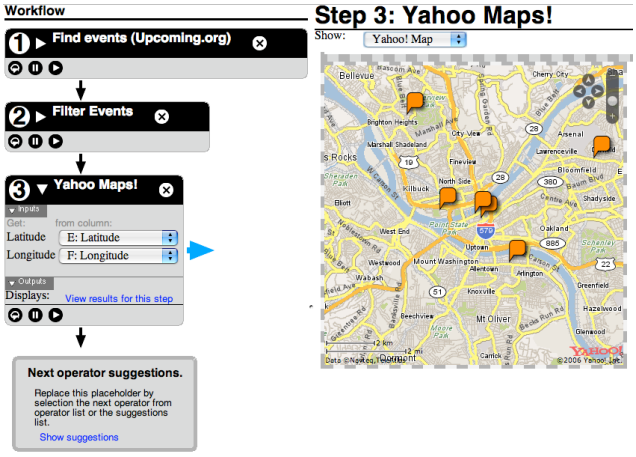

Figure 4. This figure shows the same data flow as in Figure 3 but with an extra operator added. The final operator is showing a map view rather than the default spreadsheet view.

Users also felt that it was difficult to know where to start. We modified the paper design to include template data-flows for common tasks. Some participants using our low-fi prototypes commented that it would be easier to use Marmite if they had examples to examine or modify. Template data-flows can be included to show the range of tasks that can be accomplished. This feature is not implemented in the current version of Marmite.

THE MARMITE SYSTEM

Marmite is currently implemented as a Firefox plug-in using JavaScript and XUL. We choose to make Marmite a part of Firefox rather than a standalone application for several reasons. First, we thought that close integration with a common web-browser would be beneficial for usability, adoption, and development. Being in the web browser, Marmite would be very close to the user's web browsing experience. If the user were to find some data that he or she would like to manipulate, the user would not have to launch a separate program to begin using Marmite. Second, integrating Marmite with the browser also makes it easier to manipulate private or protected data that is guarded by passwords, cookies, or other web security mechanisms that the browser is normally used to access. Third, we felt that a browser plug-in would have lower barriers to adoption than traditional applications. Installation of a Firefox extension can be done by simply clicking on a link and restarting Firefox. The Firefox platform also automatically tracks updates for plug-ins and notifies the user when new versions are available.

Marmite's interface (see Figure 1) consists of three major areas: the operator selection area (left, not shown in Figure 3 below), the data flow area (middle), and the spreadsheet display (right). Users select operators from the operator selection area, place them into the data flow, and view the current state of the data at a particular operator in a table, which shows what the data looks like after it has passed through an operator.

Data Flow View of Operations

Operators represent pieces of code that either access web services or are functions that operate locally on data. Operators are chained together in a data flow where the results of one operation are passed as input to the next operation, similar to Apple Automator [1] and Anthracite [22]. The data that flows through the operators is represented in a table. The basic unit of work for each operator is a row in the table (as opposed to columns or a single cell). Each piece of data has various attributes which are represented by different columns in the table.

Figure 3 shows an example. The first operator retrieves event information from a web service. The event objects have 6 attributes: the name of the event (column A), the date of the event (column B), the event venue (column C), the city where it is being held (column D), and latitude and longitude coordinates (offscreen). By adding objects to the data-flow, the first operator creates a schema for the data table, where each column is internally labeled with a data type. Subsequent operators add to this schema when they add columns, which might happen if additional attributes are retrieved for each piece of data.

The second operator is an event filter that removes rows from the data if they meet a certain criteria. It has a single argument; it needs to know which column to use as the date field. If the operator's inputs and the previous operator's outputs use the same label for their types, inputs and outputs can be matched automatically and connected. However, Marmite is intended to be an open platform where individuals and providers of web services create their own operators and may have different ways of labels for their data types. For example, a web service that provides event listings may have a field called "time" that indicates when an event is starting. An operator from a calendar service may call that same a field called "start time." In this case, Marmite would not automatically connect these listings service output to the calendar input; user would have to do that manually.

Some operators provide alternative displays of the data. For example, Figure 4 shows a continuation of our data flow, with a third operator that displays all of the events on top of a map. Each operator can also be collapsed to save space. For example, in Figure 4, steps 1 and 2 are collapsed.

Each operator comes with three buttons to control the execution of the operator: Reload, Pause, and Play. Operators do not execute immediately after being placed into the data flow because they may have arguments that need to be set before any meaningful execution can occur. The user then needs to press the "Play" button for the operator to begin its work.

These controls were motivated by our user study of Apple Automator. We found that users had trouble creating correct sequences of operations because the user would typically create a data flow and test it as a single unit. Unless the user had planned in advance to restrict the initial data set to a few small items, the data flow would operate on the entire set of data and errors would only be visible at the end of the process. When operations involved accesses to the network, specifically to web pages or web services, this increased the time it took to debug a process. Consequently, Marmite allows the user more control over execution, making it easier to test a data flow on a small sample of the data before committing to executing the data flow on a much larger dataset.

Operators are written in JavaScript and XBL (Extensible Binding Language). XBL encapsulates the JavaScript code for an operator, and is the glue that links JavaScript code to Marmite's operator framework. XBL makes it easy to provide some basic GUI controls and minimizes the amount of work required to interact with other operators in a data flow as well as the rest of Marmite.

Spreadsheet View of Data

In the Marmite interface, each operator is associated with a table that shows the state of the data after the operator has been run (see Figure 3). The table is displayed in the results/display area on the right. Each row in the table corresponds to a piece of data with multiple attributes. Users can view the current state of the data after executing each step. This lets the user see the effects of operations right away and identify problems immediately.

It should be noted that our spreadsheet view is currently read-only, in that end-users cannot click on a cell and modify the data. This may change in later versions of Marmite.

Inputs and Outputs

Marmite currently has three categories of operators:

> **Sources:** These operators add data into Marmite by querying databases, extracting information from web pages, and so on.
>
> **Processors:** These operators modify, combine, or delete existing rows. Example operators include geocoding (converting street addresses to latitude and longitude) and filtering. Processor operators might add or remove columns as well.
>
> **Sinks:** These operators redirect the flow the data out of Marmite. Examples include showing data on a map, saving it to a file, or to a web page.

In Marmite, each of these columns is associated with a data type that is defined by the operator that created that column. Each operator is visually divided into controls for inputs and controls for outputs. Operators that accept input from the previous operators have input parameters. Some of these may be taken from a column in the data flow and some of these may be fixed parameters that aren't specific to the row being operated on. Since data types are defined by operator authors, inputs for one operator have to be matched with the outputs for the previous operator. If Marmite cannot perform this match automatically, the user has to tell the operator where to take each argument from.

For example, Figure 3 shows the visual details of two operators. The second operator is a filter for removing rows from the data flow. It has an argument which it accepts from each row, the date of the event, and an argument which applies to all rows, the date which should be used to filter the events. The argument for the date of the event is set to column "B," which is known as "Time" to the previous operator.

Although we could have made the operators in this version of Marmite all use the same set of types, operators can be

used represent function calls to web service APIs. Web service APIs, in turn, all define their own data type. We also don't expect future operator writers to all agree on type compatibility. Although we hope to encourage operator writers to all agree on a single way to refer to data types, the design decision to allow manual type matching between operators in a data flow is based on the conservative assumption that type standardization is not guaranteed in the future.

Currently, Marmite has a small set of pre-defined data types, such as time, address, and number. Marmite also has a very basic data type resolution system. For Marmite to achieve wide-scale traction, it will require a larger set of useful data types that most authors of operators can agree on. This issue is beyond the scope of this current paper, but it is important to note that this is a fundamental and well-known problem in many domains, including federating databases, XML schemas, and the semantic web.

Operator Groupings

The operator list contains a set of operators which can be added to the data flow. But rather than using a simple list, we implemented the operator as a tree that grouped similar operators together. This was intended to help users find operators more quickly because they are able to skip over groups that are inapplicable to their task. Figure 5 shows the operator menu.

Because Marmite knows what data types are currently in the data flow, it can provide a list of "suggested operators." This list only includes operators that are compatible with the data types. We incorporated this feature to make it easier for end-users to find relevant operators.

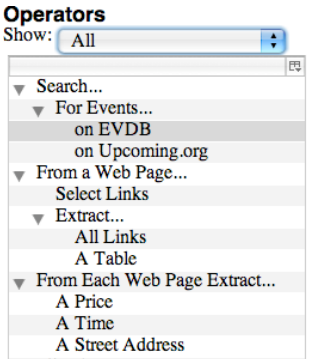

Figure 5 Operators are placed in groups to improve browsing. The menu above the list allows the user select to select alternative groupings (e.g. show only operators that can take the current data as input)

Web Page Extraction

Figures 3 and 4 show a highly specific source operator that makes it easy to extract information from a pre-specified web site, in this case, the event site upcoming.org. To extract more general information from arbitrary web pages, we also provide an operator that, when played, opens a new browser window. Users are then led through a series of wizard screens (see Figure 6) that let the user select an example of the items of interest, provide some feedback on the system's guess about the items of interest, and finally manually add or remove items that were either

Figure 6. This figure shows our wizard for selecting items of interest from a web page.

unintentionally picked or not included.

Currently, this operator only uses a simple XPath pattern matcher based on the one found in Solvent [32] to select links that might be similar enough to the user's example. Figure 6 shows a screen with the controls of the wizard laid on top. XPath pattern matching tends to work well for web pages where individual items of data in listings are in their own HTML tags.

FORMATIVE EVALUATION

We conducted a user study with 6 people to identify usability issues and how usable our design was for different classes of users. Since we incorporated some aspects of spreadsheets to make Marmite more familiar to certain types of users, we also wanted to determine how understandable these were. Note that we made two minor modifications to the user interface during the evaluation phase, as described later.

We divided our participants into three groups: users familiar with programming, users familiar with spreadsheets but not programming, and users who were not familiar with spreadsheets or programming.

We recruited participants by posting an advertisement on our local Craigslist website as well as a popular university bulletin board. Using email surveys, we asked our

participants about Internet use and familiarity with spreadsheets and programming. We had two people in each of the following groups:

- Little or no spreadsheet or programming familiarity ("no-experience" group)
- Familiar with spreadsheets (including formulas)
- Familiar with programming

Participants who had taken programming classes in the past but claimed to be bad or unfamiliar with programming were not included in the programming group.

The testing was conducted on a 12-inch Apple Powerbook G4. A mouse was provided and was used by most users.

Tasks

Because Marmite is not intended as a walk-up-and-use system, users were provided with a warm-up task to help familiarize them with the system. The instruction sheet for the warm-up task explained what actions were needed to retrieve a set of addresses and how to geocode an address (that is, transform a street address into latitude and longitude, a prerequisite before putting items onto a map).

After completing the warm-up task, users were asked to complete three more tasks of increasing complexity. The three tasks were:

1. Search for events and filter out events further than a week away. Here, we wanted to see if participants could use two operators to achieve a result, and to assess the basic usability of the system.
2. Compile a list of events from multiple event services and plot them on a map. (The two event services have different output schemas, can users make sure they are merged?)
3. Given a web page with links to some apartment rental listings, plot those apartments on a map. We wanted to understand if our design for screen-scraping links from a page was usable. This is similar to the functionality offered by the mashup HousingMaps.com. Figure 7 shows an example of a completed data flow for this task.

Our first two participants were unable to do most of the tasks due to labeling problems in the operators. We added some labeling to indicate that the column-selection menus would tell the operator which column should be examined for each argument that it needed. Of the four remaining users, both users in the programming group and one user in the spreadsheet group were able to complete the tasks with little difficulty.

We also decided to turn off the automatic appearance of suggested operators after the first test when the user got extremely disoriented by apparently unexplainable changes. We changed this so that instead of appearing automatically, user would have to ask for them.

Observations

Three of the participants were able to complete most of the tasks without difficulty: one from the spreadsheets group and two from the programming group. Participants who were able to complete all of the tasks completed them in under an hour.

The users in the programming group seemed enthusiastic about the tool and wanted to be notified when Marmite was complete. One user mentioned that it would be possible to use a tool like this to replicate the some of the functionality data aggregation and visualization services such as the

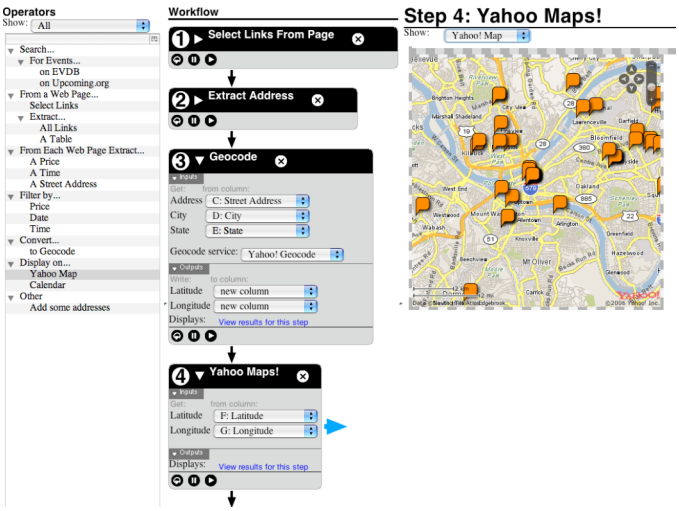

Figure 7 A dataflow that mimics the functionality of the housingmaps.com website. Note that Step 1 and 2 are collapsed to save space.

Multiple Listing Service real estate database, which he remarked was too expensive for ordinary people.

If users were to do this in a programming environment, they would have to figure out how to access web services represented by our operators or replicate the code they represent, write the code, and debug it. Furthermore, this would have been impossible for the users in the spreadsheet group. It is also worth noting that the housingmaps.com mashup replicated in this task was developed by a highly skilled professional programmer.

Since the focus of our user study was to identify usability problems in our current tool, the rest of this section discusses problems we uncovered.

For the 3 participants who were unable to complete the tasks, the main barrier was understanding the concept of a data flow. These users were puzzled by the meaning of selecting inputs and outputs. Users deleted operators they had successfully used not knowing that they were erasing data. These users believed that an operator was no longer needed once it had been used to produce a result in the spreadsheet view. They also did not seem to believe that there was any particular order to the operators, viewing them as highly interactive tools to simply change the

current state of the data, not as pieces of a sequence that needed to be built up.

For the most part, users had several did not attempt to have the same model of how the operators interacted with the results tables that were displayed on the right. When these users played an operator, they believed that operator has made some changes to that copy of the table. In our system, each operator has its own copy of the data, which is present in the third column. Deleting an operator would delete the table associated with that operator. The operator after it in the data flow would have no prior operator to fetch the data from. Similarly, users who were stuck would close operators and lose all of the results from those operators. One possible solution is to change operators so that they do not own tables and instead to make tables objects in their own right.

One of the non-programming users did not like trying to accomplish their tasks with primitive operators. She said that they preferred richer interactions where, once the data type was introduced to the table, helpful displays about that data should made visible. For example, one user commented that once he extracted a list of URLs to web pages that each had rental listings, he would have liked to have immediately other useful attributes such as price, location to be extracted without having to use separate operators to get each piece of data.

FUTURE DIRECTIONS

Our user tests revealed a number of directions for improvement:

In this version of Marmite, we only had a very basic screen-scraping operator. We plan add more sophisticated screen-scraping functionality and incorporate a more diverse set of heuristics and data detection algorithms.

Another direction that we are looking at is to significantly expand the set of operators, and create a way to transform a data flow into a mashup website. The recent explosion of mashup-making activity was caused by the increased availability of web services APIs. A similar increase in the number of available operators might encourage experimentation. One method of doing this is to automatically generate operators from machine-readable web service descriptions (WSDL) offered by web services providers.

We also plan to make some changes in operator feedback or the placement of the data tables on the screen to address the usability problems encountered by some of the non-programmer users.

CONCLUSIONS

It is important to note that there is a growing need for a tool like Marmite. Existing web sites and cannot always predict the needs of all of its end-users. Thus, it is important to provide tools that can help end-users help themselves.

In this paper, we presented the design, implementation, and evaluation of Marmite, a tool for end-user programming on the web. Marmite works by displaying a linked data flow / spreadsheet view, letting people see the program as well as the data simultaneously. A small user study showed it was possible to replicate the functionality of a popular mashup website.

ACKNOWLEDGMENTS

This work is supported in part by a National Science Foundation grant IIS-0646526 and Intel Research. We also thank Duen-Horng (Polo) Chau for his support and assistance. We thank Michael Twidale and Cameron Jones for their enthusiasm and support.

REFERENCES

1. Apple, Automator: Your Personal Automation Assistant. http://www.apple.com/macosx/features/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. Cohen, W.W., Minorthird: Methods for Identifying Names and Ontological Relations in Text using Heuristics for Inducing Regularities from Data. 2004. http://minorthird.sourceforge.net
5. Cypher, Allen, ed. "Watch What I Do: Programming by Demonstration", MIT Press, Cambridge MA, 1993.
6. Dapper. http://www.dappit.com
7. DataMashups. http://www.datamashups.com
8. eGrabber, AddressGrabber Business 3.2.1.
9. Faaborg, A. and Lieberman, H.. A goal-oriented web browser. *Proc. CHI 2006.* ACM Press (2006) pp. 751-760
10. Fujima, J., A. Lunzer, K. Hornbaek, and Y. Tanaka. Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access. In Proc. UIST2004, CHI Letters 6(2). ACM Press (2004). pp. 175-184.
11. Huynh, D., Mazzocchi, Stefano and David Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. International Semantic Web Conference (ISWC), November 2005, Galway, Ireland.
12. Huynh, D. F., Miller, R. C., and Karger, D. R. Enabling web browsers to augment web sites' filtering and sorting functionalities. *Proc. UIST '06.* ACM Press (2006), 125-134.
13. 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.

14. Kistler, T. and Marais, H. WebL - a programming language for the Web. Proc. WWW7, (1998) 259-270.
15. 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, pp. 1-4
16. Ko, A. J., Myers, B. A., and Aung, H. Six Learning Barriers in End-User Programming Systems. IEEE Symp. On VLHCC, (2005) pp. 199-206
17. Ko, A. J. and Myers, B. A. Designing the Whyline: A Debugging Interface for Asking Questions About Program Failures. *Proc. of CHI 2004*, ACM Press (2004), 151-158.
18. Kuhlins, S. and R. Tredwell, Toolkits for Generating Wrappers - A Survey of Software Toolkits fo Automated Data Extraction from Web Sites. LNCS 2591 2003: p. 184-198.
19. Laender, A.H.F., B.A. Ribeiro-Neto, A.S.d. Silva, and J.S. Teixeira, A Brief Survey of Web Data Extraction Tools. SIGMOD Record 2002. 31(2): p. 84-93.
20. Lieberman, H. (Ed.) 2001. Your Wish is My Command: Programming by Example. San Francisco: Morgan Kaufmann.
21. H. Liu, P. Singh, ConceptNet — A Practical Commonsense Reasoning Tool-Kit, BT Technology Journal, *22,* 4, p.211-226, October 2004
22. Metafy Anthracite. http://www.metafy.com/products/anthracite
23. Microsoft, Microsoft Office XP Smart Tags. http://www.microsoft.com/technet/prodtechnol/office/officexp/maintain/xptags.mspx
24. Miller, R.C., Lightweight Structure in Text, Unpublished PhD thesis, Carnegie Mellon University, Pittsburgh, 2002.
25. Miller, R. and Bharat, K. SPHINX: A Framework for Creating Personal, Site-Specific Web Crawlers. Proc. WWW7, (1998), 119-130.
26. Mozilla Foundation, *XBL (Extensible Binding Language).* http://www.mozilla.org/projects/xbl/xbl.html
27. Mozilla Foundation, *XUL (XML User Interface Language)*, http://www.mozilla.org/projects/xul/
28. Myers, B. A., Pane, J. F. and Ko, A. Natural Programming Languages and Environments. Comm. Of the ACM, (Sept. 2004), 47-52.
29. Nardi, B.A., J.R. Miller, and D.J. Wright, Collaborative Programmable Intelligent Agents, Communications of the ACM, vol. 41(3): pp. 96-104, 1998
30. Pandit, M.S. and S. Kalbag. The Selection Recognition Agent: Instant Access to Relevant Information and Operations. In Proceedings of International Conference on Intelligent User Interfaces. Orlando, FL. pp. 47-52 1997.
31. schraefel, m.c., Modjeska, D., Wigdor, D., and Zhu, Y. Hunter Gatherer: Interaction support for the creation and management of within-Web-page collections. Technical Report CSRG-437, Department of Computer Science, University of Toronto, October 2001.
32. Solvent, http://simile.mit.edu/solvent/
33. Sugiura, A. and Koseki, Y. Internet scrapbook: automating Web browsing tasks by demonstration. Proc UIST '98. ACM Press (1998), 9-18.
34. W3C, *XPath (XML Path Language)* http://www.w3.org/TR/xpath
35. W3C, *Semantic Web*. http://www.w3.org/2001/sw/