End-User Programming of Mashups with Vegemite

James Lin[1], Jeffrey Wong[2], Jeffrey Nichols[1], Allen Cypher[1], and Tessa A. Lau[1]

[1]IBM Almaden Research Center
650 Harry Rd
San Jose, CA 95120 USA
{jameslin, jwnichols, acypher, tessalau}@us.ibm.com

[2]Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
jeffwong@cmu.edu

ABSTRACT
Mashups are an increasingly popular way to integrate data from multiple web sites to fit a particular need, but it often requires substantial technical expertise to create them. To lower the barrier for creating mashups, we have extended the CoScripter web automation tool with a spreadsheet-like environment called Vegemite. Our system uses direct-manipulation and programming-by-demonstration techniques to automatically populate tables with information collected from various web sites. A particular strength of our approach is its ability to augment a data set with new values computed by a web site, such as determining the driving distance from a particular location to each of the addresses in a data set. An informal user study suggests that Vegemite may enable a wider class of users to address their information needs.

Author Keywords
End-user programming, programming by demonstration, mashup, data integration, automation, web

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

INTRODUCTION
Mashups are applications that combine functionality and data from multiple web sites to help users solve tasks not originally envisioned by the web site designers. Unfortunately, constructing mashups typically requires programming skill, which limits non-programmers to using mashups that have been created by others. To address this problem, designers and researchers are building tools that allow users to create mashups without programming, with some success. Often, the focus in these tools is on building mashups that are robust to changes in their data sources and can be reused by many users at some time in the future.

However, there is a whole class of *ad hoc mashups*, where users want to quickly combine data from multiple web sites in an incremental, exploratory fashion, often in support of a one-time or infrequently performed task. In this case, the collected data is the key artifact, not the code that extracted or combined the data. Current mashup tools do not always adequately support ad hoc mashups, because they encourage users to focus on the mashup's code instead of the data. In some cases, robustness and reuse are favored over helping users perform their tasks.

To support end-user creation of mashups, we have extended the CoScripter end-user programming system (formerly Koala) [14] to allow users to collect and process data across multiple web sites. Using this extension, called *Vegemite* (see Figure 1), users start with an empty table, called a *VegeTable*. Data can be manually entered into the VegeTable, copied from an existing source like a spreadsheet, or extracted from an existing web page using our own direct-manipulation tool. Users then demonstrate, through a series of actions on the table and the web, how to fill additional columns into the table. These actions are recorded into scripts, which can be re-executed immediately for other rows in the table and used later to refresh the data in the table. For example, a user might create a script that uses Yahoo! Maps to determine the driving distance between an address stored in one column of the table and her place of employment. This might be useful for prioritizing a list of houses for sale based on the length of her commute. The reusable script is created automatically simply by watching the user demonstrate how to get the driving distance from Yahoo! Maps based on the data in one row of the table.

The use of scripts is an important difference between Vegemite and previous tools. Most existing mashup tools, even those targeted at end users and for creating ad hoc mashups, focus on one particular process: separately extracting tables of data from multiple web pages and joining them together based on common values. Vegemite's use of scripts works best for a different process: collecting a table of data from anywhere and then running scripts that add columns to that table based on the data in each row.

Another key feature of Vegemite is that the scripts can do more than just collect data. Vegemite scripts, like the CoScripter scripts they are based upon, contain arbitrary low-level web actions, such as clicking on links and entering values into form elements. As a result, scripts can also be used to perform actions based on values in the table. For

IUI'09, February 8–11, 2009, Sanibel Island, Florida, USA.

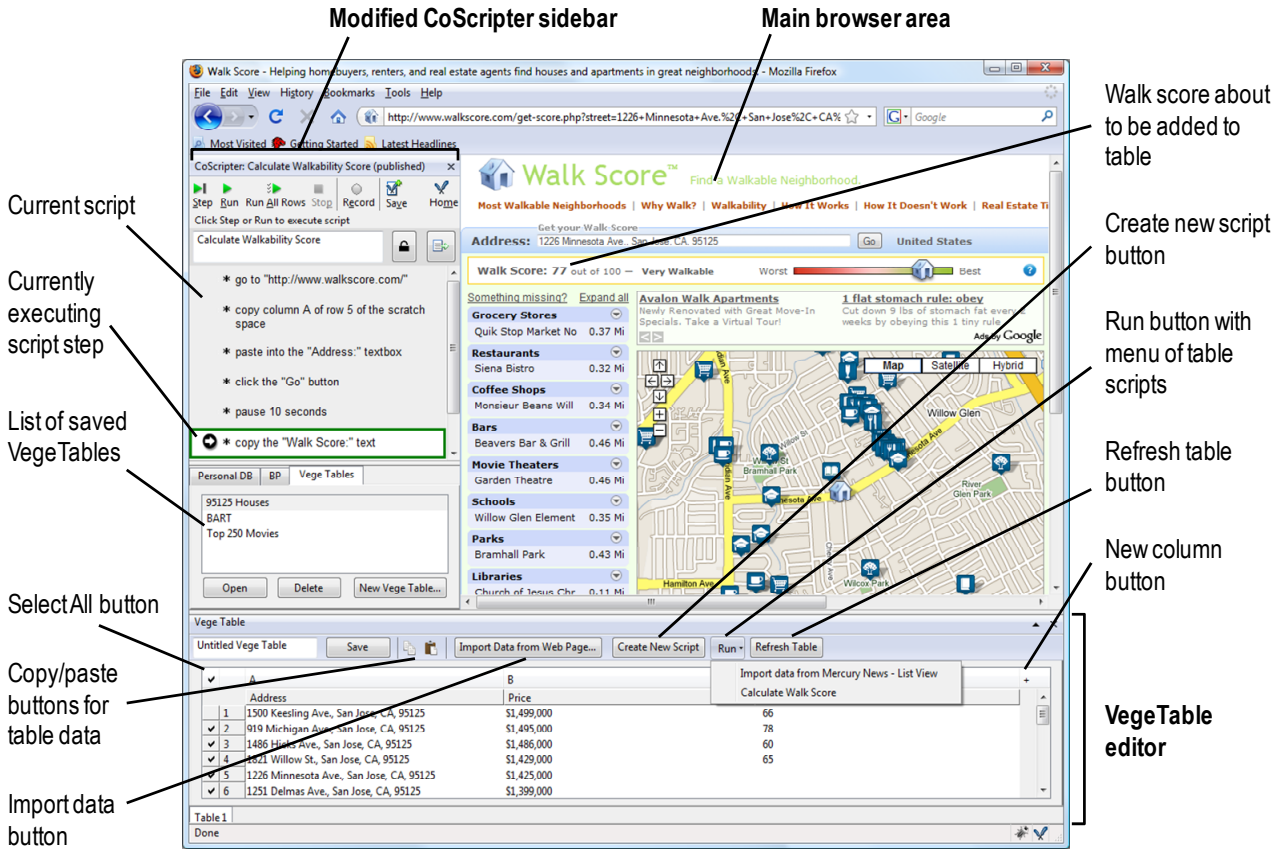


Figure 1. The Vegemite user interface.

example, values in the table might be included in an e-mail sent by a script that automates a web-based e-mail application. We believe Vegemite is the first system that enables end users to use programming-by-demonstration to both aggregate data and perform web-based operations based on the resulting data.

There are advantages and disadvantages to Vegemite's process. An important advantage is that scripts are created and executed in the context of web browsing, which all internet-savvy users already understand. Furthermore, Vegemite supports a mixed-initiative interaction style, where the system can automate portions of the browsing process but allow the user to step in to fix errors as they occur. The main disadvantage is that each script must be executed for each row in the table, which can be time consuming, depending on the number of rows in the table and the number of steps in the script.

We begin by putting Vegemite in context with the large body of work in the space of mashups. We then describe the design of the Vegemite system, discuss its benefits and limitations, and then describe a user study that we have conducted with the system.

RELATED WORK

A key component of Web 2.0 technology has been the idea that web data and tools should be made accessible through standard web service APIs. Developers have capitalized on this availability of data by combining multiple data streams and making use of easily available visualization tools, such as Google Maps. Although originally these mashups could only be created by web-savvy developers, end-user tools and platforms have begun to emerge that lower the barrier to creating mashups. A comprehensive list of mashups and tools can be found at http://programmableweb.com.

Mashup tools often focus on one or more of the following tasks:

- *Extracting data from existing data sources*, such as web pages, web services, and feeds.
- *Aggregating, or "joining," data from different sources into a single data set.* Data from each of the sources needs to have at least one common field, such as the name of a restaurant, so that records from one set can be matched with their corresponding records in other sets.
- *Visualizing the data in a way that allows the user to understand the aggregated data.* Many mashups that

include geographic data, such as addresses, are visualized using Google Maps.

Data extraction is a key feature for many existing mashup tools, and the method of extraction is an important differentiator among these tools. Some tools can only extract data from structured data sources, such as web services, RSS feeds, or documents annotated with RDF. Other tools work with unstructured data sources, such as the human-readable information on normal web pages. Vegemite is a member of the latter category as it focuses on extracting data and executing operations on normal web pages.

Mash Maker [5] supports extraction, aggregation, and visualization across web sites. Before an end user can extract data from a web page, an extractor must be created for that web page. The interface for doing so is mostly direct manipulation but requires some technical expertise, such as understanding URL arguments and regular expressions. Mash Maker's user model is that expert users will create extractors and that other users will leverage them. In contrast, Vegemite is designed around one user of low to moderate skill that does both extraction and aggregation.

To combine two web sites that have extractors with Mash Maker, end users extract data from one site by interactively "copying" it, and then aggregating that data by "pasting" the extracted data into the other site. Mash Maker guesses how the data may relate based on the copy and paste operations that the user has performed, and provides an interface that allows the user to correct any mistakes that the system has made. Mash Maker uses a URL-based scheme for parameterizing the mashups that it creates. This means that the pages that data is copied from must have an understandable URL which Mash Maker can modify to make reasonable queries for other values. In contrast, Vegemite scripts explicitly define the query in terms of the browser operations that are required for a user to perform the query. This is independent of the means that the web site uses to communicate its data, such as Ajax or HTTP POST requests that do not include parameters in the URL.

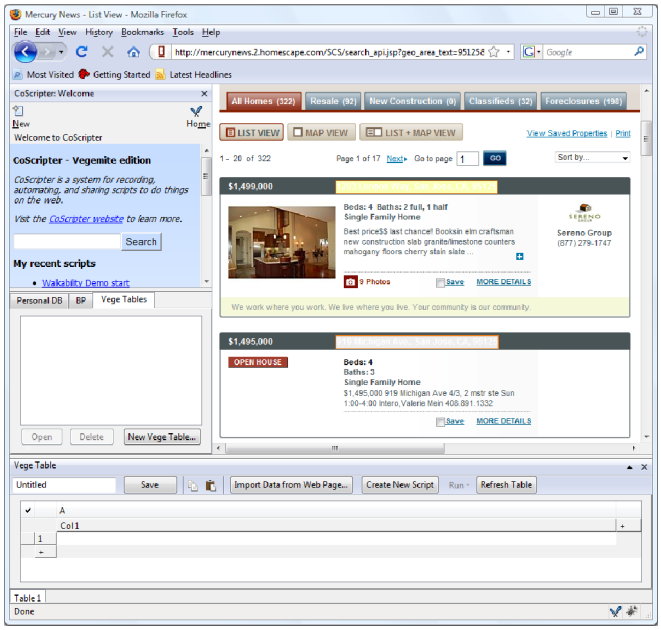
Karma [19] lets users extract data from a web site into a data table through demonstration. To extract data, users visit a web page and click on the pieces of the page they would like to extract, such as a restaurant name and address. Karma uses an XPath generalization scheme to find similar data on the same page and other related pages, and then copies this data into a table. This extraction process is similar, though more advanced, than Vegemite's bulk data extraction feature. Karma also keeps a repository of extracted data tables and can suggest relations between tables, enabling data integration. Karma's strength is in extracting and combining tables of data from existing web sites, but it does not support arbitrary operations on tables, such as executing queries based on a table or collecting data from computations performed on the data in a table.

The SIMILE effort at MIT has several mashup projects, which use both structured and unstructured data sources. Sifter [10] and Solvent [18] are tools for end users to extract information from an unstructured HTML page into a structured data table, through a point-and-click interface. Piggy Bank [9] is a Mozilla Firefox extension that detects RDF data within web pages, either embedded in the web page or scraped via a Solvent screen scraper, and stores it so that the user can interact with the structured data. Potluck [11] is a tool for easily combining web pages with embedded structured data sets into a single set, with novel interaction techniques for cleaning up and aligning data between sets. Like Karma, these projects focus either on data extraction into tables or joining data from multiple tables together. None of these tools provides support for adding new values to their data sets through computations performed on other web sites.

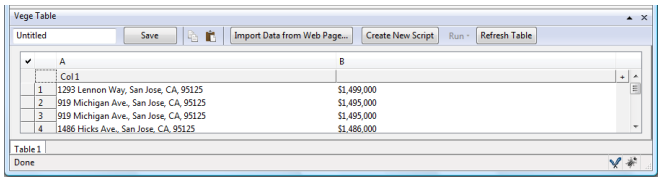
Summaries, Relations, Cards, and Templates [3, 4] (SCRT) is a framework for users to extract information from a web site into a structured dataset, associate it with related data from other web sites, and lay out the combined information in a set of cards. This work also uses an XPath-based scheme to extract data, much like Karma, Sifter, and our own bulk data extraction method. A user who has collected data with SCRT augments the set through the use of a search engine on other sites that SCRT has already extracted data from. This goes beyond previous work by supporting a particular kind of web query to expand the data set. However, Vegemite goes further by supporting the use of any web query that can be represented in our scripting language.

Other mashup tools often involve visual programming languages [16, 21, 22] or GUI assembly tools [7] where widgets representing the different web sites or web services are laid out on a web page and connected together with some programming language. Another approach is to connect disembodied forms of input fields of different web sites together using formula languages from spreadsheets [5, 7]. In these cases, the focus is on data aggregation, and extraction is performed by pre-programmed code modules. Often the data used by such tools is extracted from structured data sources, such as RSS feeds and web services.

The user interfaces of these aggregation tools also create a focus on the operations that aggregate data, rather than the data itself. This focus can get in the way of the user's goals. For example, in a user study of the Marmite [21] mashup system, users added operations to a dataflow. Each operation generated a table that showed the state of the data as it was processed by the operation. Once the operation was complete and the data appeared to have been processed correctly, non-programmers discarded the operation because it had served its purpose. However, discarding the operations also discarded the data, to the users' surprise. Users were more comfortable with using operations as *tools* to transform the data rather than as *steps in a process* for a computer to execute.



a)



b)

Figure 2. Extracting initial data for the walkability scenario. a) Vegemite in bulk extraction mode on the Mercury News site. b) The data after being extracted into the VegeTable.

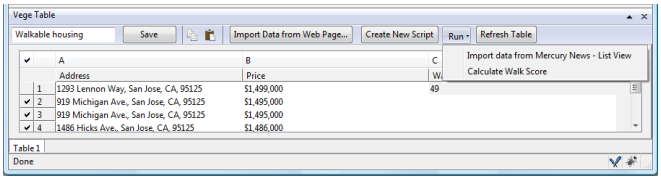


Figure 3. The Vegemite interface after the "Calculate Walk Score" script has been recorded. Selecting the script from the Run menu will run the script for each of the selected rows.

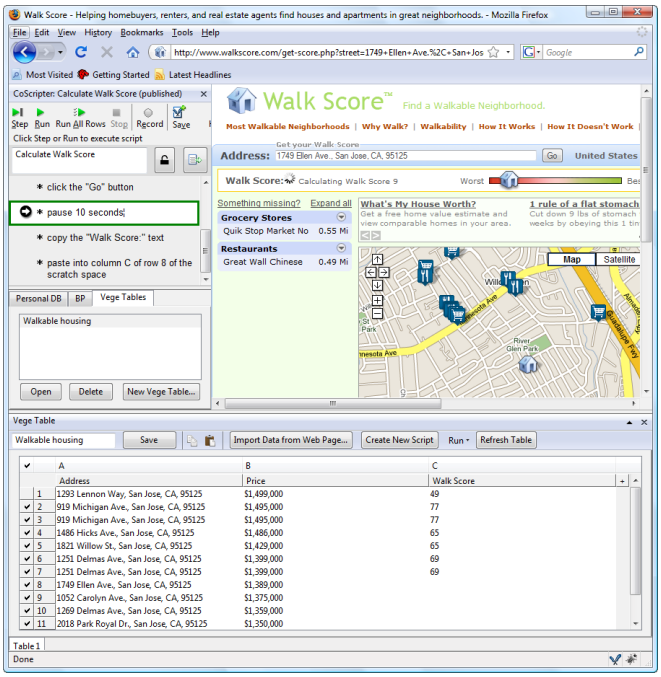


Figure 4. Vegemite during its computation of the walk scores for all of the remaining rows in the walkability scenario.

Most existing mashups are web sites that present users with a web interface to data and services. These sites were designed by programmers with the goal of being used repeatedly by many users. Many of these mashups can be thought of as a search component coupled with a visualization element that gives another view on the data. Creation tools for these "dashboard-style" mashups (e.g., IBM Lotus Mashups [12]) require users to lay out widgets representing the web sites to be included in the mashup. For these tools, the focus is on laying out widgets, not on the data itself.

A few tools have been created that allow users to use existing web sites as tools to perform computations. Clip, Connect, Clone for the Web (C3W) [7] allows user to clip elements from existing web pages and use them together to create useful combined applications. Unlike Vegemite, C3W interfaces cannot be integrated with a data table or automatically be applied to all rows of such a table.

Transcendence [1] allows users to extract large amounts of data from the "deep web" by automating many submissions of the same form on a web site and allowing users to collect all of the results into a single page. Vegemite provides a more general system for scripting the web to perform the queries like those possible with Transcendence, and Vegemite also allows this data to be aggregated with data collected from other web sites.

VEGEMITE'S USER INTERFACE

Vegemite is implemented as an extension to the Mozilla Firefox web browser. Its user interface consists of two main parts: a sidebar and a VegeTable editor (see Figure 1). The sidebar along the left, borrowed and modified slightly from CoScripter [14], is for creating, executing, and saving scripts. It also contains a list of VegeTables that the user has stored, and has buttons for opening saved VegeTables and creating new ones. The editor along the bottom of the browser window consists of a spreadsheet-like table for storing data, and buttons for extracting data and running scripts over the table.

An example of using Vegemite

Jane is looking to buy a house in San Jose, California and has been using the real estate listings in the San Jose Mercury News web site to search for houses. One of her main

criteria is whether a house's neighborhood is "walkable," i.e., within walking distance of shopping, jobs, and transit. She finds a web site called WalkScore.com that takes an address and calculates a walkability score. Jane would like to calculate scores for each house in a list of those currently on the market.

Jane first navigates to the real estate listings section of MercuryNews.com, enters her criteria (such as ZIP code and number of bedrooms) and clicks Search. After getting the results, Jane decides to import this data into a VegeTable so that she can augment the data with walkability scores.

Jane opens the CoScripter sidebar, clicks on the VegeTables tab at the bottom, and clicks "New VegeTable…" This opens the VegeTable editor along the bottom of the browser window. She then clicks the "Import Data from Web Page…" button, which starts Vegemite's bulk data extraction mode. Once in this mode, Jane can click on the web page elements that she wants in one row of her table. In this case, she clicks the address and the price for the first house in the list of results. As she clicks on the address for the first house, Vegemite highlights that element, and then also highlights the addresses of the other houses in gray, indicating that these addresses will be extracted into the first column of the table (see Figure 2a). She also clicks on the price for the first house, and Vegemite highlights the corresponding prices for the other houses in the search results. To finish extracting, she clicks "Done Extracting." Vegemite takes the extracted data and adds them to the table (see Figure 2b).

Now Jane wants to add a column to her table and populate it with walkability scores. First, she adds a new column in the table and labels it "Walk score." She then clicks on the "Create New Script" button at the top of the editor, which starts recording a new script for adding a walkability score to a row in the table. She then navigates the browser to WalkScore.com, copies the address from the first row in the table, pastes it into the textbox in the web page, and clicks Search. After the result appears, Jane copies the score from the page and pastes it into the "Walk score" column of the first row. Vegemite records all of Jane's actions into a script whose language resembles English, and displays the script in the CoScripter sidebar. She clicks Save in the sidebar and names her new script "Calculate Walk Score." Vegemite automatically associates this script with the currently opened VegeTable, which adds the script to the VegeTable's "Run" menu (see Figure 3).

Jane can now use her script to fill in the walkability score for the rest of the rows. She clicks the button with the checkmark in the upper left-hand corner to select all of the rows, goes to the Run menu, and chooses "Calculate Walk Score" (see Figure 3). Vegemite loads the script and runs it once for each selected row, replacing instances of "row 1" in the script with the current row number, thus calculating the walkability score for every row (see Figure 4).

Scripts can also help the user perform operations across the collected data. For example, once Jane has collected all of the walkability score data, she decides to e-mail the five houses with the best scores to her real estate agent. So she logs into Gmail.com and composes a new message. She creates a new script called "email listing" where she copies the address and price of the house with the best score and pastes it into the Compose box. She then selects the other four houses and runs the "email listing" on those rows. She finally clicks Send. Of course, such a script would not be useful for sending a list of walkable houses just once, but this script might be useful for sending a new list of houses to the real estate agent every week.

Other Scenarios

The above scenario is representative of Vegemite's strength, which is using existing web sites to compute new values for a data set. The following scenarios demonstrate other uses of the system.

Vegemite can be used to find a list of the closest yogurt shops to a particular address in terms of driving distance. To create this mashup, the user would extract a list of yogurt places from a web site such as yellowpages.com, and then record a script that visits a map web site and calculates the driving distance. The table of yogurt shops and distances can be used to make a decision about which restaurant to visit.

The U.S. Department of State publishes a monthly bulletin that lists which visa numbers have been processed in the past month. Aggregating the information from all of the archived bulletins can allow waiting applicants to determine how fast the visas are being processed on average. This information can be extracted in Vegemite by creating a table from the list of all available archives and then creating a script that browses into each archive and extracts the necessary values.

The Internet Movie Database (IMDb) publishes a list of its top 250 movies as voted on by visitors to the web site. Vegemite can be used to extract this table and then annotate it with other information about each movie, such as the name of each movie's director.

ARCHITECTURE

Vegemite consists of two main parts: VegeTables for storing data, and the CoScripter engine for recording and playing back actions on web pages.

CoScripter

CoScripter [14] is an extension to the Mozilla Firefox web browser that allows users to record and play back web-based processes. As users interact with the web browser while CoScripter is in recording mode, their actions are recorded into a pseudo-natural language script. The scripting language's syntax resembles English, so that people can understand it in addition to computers. Saved scripts are

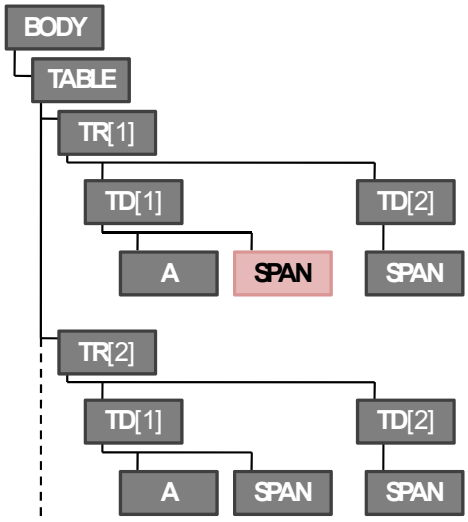


Figure 5. An example of DOM structure from a web page with repeating content, such as search results. The highlighted SPAN node indicates the element that the user selects first after entering bulk extraction mode.

stored in a central repository where they can be viewed and executed by other users.

Vegemite extends the CoScripter engine with several capabilities: copy and paste operations, recording and playing back operations on the table, and selecting strings from within a web page. We have also added several user interface elements in the sidebar to support interactions with the VegeTables.

VegeTables

A VegeTable consists of a set of tabular data and scripts that extract data into the table or perform operations on the table data. VegeTables are stored on a central server, which is the same server that stores CoScripter scripts. This enables users to access their data on any machine with Vegemite installed. In the future, storing the tables on a web server will make possible sharing VegeTables with other Vegemite users and using them as inputs to other mashups.

EXTRACTING DATA IN BULK

We realized early on that Vegemite's script-based nature works best if the user already has a data set available on which the scripts can be run. In order to support the use of scripts, we added a bulk data extraction feature to Vegemite that allows users to populate the VegeTable with a large quantity of data from a single web page. Once the table is populated, scripts can easily be applied. The bulk data extraction algorithm that we have implemented is similar to the XPath-based schemes employed by Sifter [10], Karma [19], and Dontcheva et al's systems [3, 4].

Our bulk extraction mode is an interactive process. When in extraction mode, elements on the current web page will highlight as the user moves the mouse above them. The user's goal is to click on the elements that represent data that should be extracted into the first row of the VegeTable. When the user clicks an element, she is providing an example of the data that should be included in one column of the VegeTable. Vegemite uses all of the clicked elements to determine the rest of the data that should be extracted from the page.

Our extraction algorithm makes use of the tree-based DOM representation that all web pages are constructed from. An example of a DOM tree for a page with repeating content is shown in Figure 5. In this particular page, each result is displayed in a single row made up of two table cells. Suppose that the first element that the user clicks on during the extraction process is the SPAN element highlighted in Figure 5. In order to determine the other data to extract, Vegemite first determines the deepest node in the tree that is a parent of all of the clicked elements. Since only one node has been clicked, this common parent is just the parent of the SPAN node.

For the parent node, we compute a partial XPath expression that describes how to find the clicked nodes from the parent. We then search for siblings of the parent node that have the same type, and use the partial XPath expression to determine if those siblings have child nodes in the same location as the node that the user has clicked. All matches are counted. The process is then repeated for the parent node's parent until we reach the BODY element of the document. Match counts are saved for each parent, and the parent receiving the highest match count is used as the basis for extracting data.

In the particular case of Figure 5, we would start by computing matches for the parent of the SPAN, which we can describe as TR[1]/TD[1]. The match count for this parent is 1, because TR[1]/TD[2] also contains a SPAN element. We then proceed to the parent of TR[1]/TD[1], which is TR[1]. The match count for this parent is at least 1 as shown in the figure, because TR[2]/TD[1] contains a SPAN element. The dashed line in the figure is meant to suggest that more rows are in the document than are shown in the figure, and we can assume that the match count would be approximately equal to the number of rows in the page. As we continue up the tree, the TABLE parent will have 0 matches and then the algorithm completes.

Each time the user clicks another element, the algorithm is run again to determine a new hypothesis for the elements that should be extracted starting from the common parent of all the clicked elements. We do not include the previous hypothesis in the determination of the next hypothesis. This means that the data set that Vegemite would extract can change radically as the user selects additional elements; however, this does not seem to happen in practice.

Once the user is done clicking elements, she may click the "Done Extracting" button to extract data into the table. For the parent with the highest match count, the parent element and each of its siblings that contain data are extracted into the table as a separate row. Some rows may be lacking data for some columns, and in such cases those cells will be left

empty. Any row that contains no data for any column is not included in the table.

OPERATING ON TABLES WITH SCRIPTS

Once users have extracted a data set into a VegeTable, they can use Vegemite's scripting capabilities to augment the data and create operations that act on the data. Vegemite extends CoScripter's scripting language, which is capable of executing actions that a person can perform in a web browser. A typical CoScripter script contains statements such as click the "Go" button or enter "washington" into the "Search" textbox. When CoScripter runs a script, it literally performs the statement's action in the web browser.

Vegemite adds the following capabilities to CoScripter:

- recording and playback of copy and paste interactions
- recording and playback of operations that modify the VegeTable
- generalization of row labels for applying scripts to different rows than they were recorded on
- labeling for certain non-form elements
- identification of any page element via XPath

The standard copy and paste operations are used by Vegemite as its principal means of moving data from web pages to the VegeTable and from the VegeTable back into web pages. We chose to use copy and paste because they are operations that users are already familiar with and can easily demonstrate to the system. In contrast, most other mashup systems use special DOM-based selection mechanisms to extract data from content. Mash Maker [5] does make use of the copy/paste metaphor as a part of this process, but it implements its own set of copy/paste operations, whereas Vegemite records and plays back the built-in functionality available in Firefox. Using the built-in functionality allows us to record copy/paste in the VegeTables in exactly the same way we record copy/paste for web pages.

We have also added recording and playback of the bulk data extraction process. CoScripter scripts may now contain lines such as begin extraction and end extraction. These lines work with the standard CoScripter recording and playback of clicks within web pages to allow for recording and playing back the entire bulk extraction process. Scripts can record and play back most modifications that happen to the VegeTable.

Actions recorded on the table must reference the appropriate table cell, for which we use the format column D of row 1. Scripts are currently recorded in the context of whatever row the user chooses for demonstration. If the user pastes a value into row 1, then row 1 will be recorded. If the user uses row 5, then row 5 will be recorded. However, when the user selects a row in the VegeTable and runs a script, Vegemite modifies the row numbers in the script to match the selected row. This is similar to the way cell references in spreadsheets are changed when they are copied into different cells. Vegemite then runs through the script step by step so that the user can watch Vegemite perform the actions of the script on each row. When the user selects multiple rows and runs a script, Vegemite performs the substitution as each row is processed.

When we added support for copy and paste from web pages, we discovered that CoScripter's existing labeling system was not sufficient to label all of the target text that might be copied. The original labeling system was designed to find labels for interactive elements, such as form elements and links, and heuristics such as using the text to the left of the copied selection were not always effective. We added some additional heuristics to improve the labeling system. For example, if the text being copied is within a table cell, Vegemite uses the row and column headers of the cell as labels.

In a few instances, we found that the labeling heuristics were not sufficient. In these cases, when a human-readable label cannot be found, we fall back to using an XPath expression that identifies the location of the element.

Reusing Vegemite Mashups

We designed Vegemite to support the creation of ad hoc mashups: an aggregation of data that is collected once, used for some purpose, and then thrown away. However, even if the data is not useful again, it may be useful to reuse the operations used to extract and aggregate the data. From CoScripter, we already have the ability to save the scripts which are the principal means of aggregating new data in Vegemite. We have also added features to CoScripter that allow us to save the bulk data extraction process as a script.

A key addition to the saving of scripts is the linking of scripts with a VegeTable. Each table not only saves its data, but also the scripts that were used to extract and aggregate that data. We have built in several interface features that allow users to make use of their saved operations on a VegeTable (see Figure 1). The Run menu allows users to regenerate the values of particular columns. We have also added a "Refresh Table" button that will run all of the scripts saved on a table in the appropriate order, thus regenerating the data in the table with more recent values.

FORMATIVE USER STUDY

We conducted an informal user study of Vegemite to test the ease of use of the interface, and to find out whether users would be able to apply Vegemite in a variety of situations.

Eight volunteers from our research lab participated in the study. Four were female, four were male, and six had significant programming experience. Two of the programmers had previously created mashups on their own. Participants were compensated for their time with a US$10 lunch coupon for our lab cafeteria. The study took approximately 60 minutes to complete.

All study sessions were conducted in an office at our research lab using a laptop computer that had the latest version of Firefox and the Vegemite extension. Each session began with the experimenter giving the subject a short demo of using

Vegemite to create a simple mashup of the IMDb Top 250 Movies list with the names of the movie's directors from elsewhere on the IMDb web site. This demo took approximately 5–10 minutes.

After the demo, the subjects were asked to construct three mashups on their own. These three mashups correspond to three of the scenarios for Vegemite that we described earlier:

- **Real Estate Walkability:** For all of the houses for sale in the 95003 ZIP code with asking prices between $1,750,000 and $2,000,000, you want to know their Walk Score. This task required use of the following web sites: www.mercurynews.com/realestate and www.walkscore.com.
- **Nearby Yogurt**: For all of the yogurt places in the 94301 ZIP code, find the driving distance from the house address of 680 University Ave, Palo Alto, CA. This task required use of the following web sites: www.yellowpages.com and maps.yahoo.com.
- **Visa Bulletin Dates:** For all of the archived Visa Bulletins, extract a list of the dates they show for the "Employment-based 3rd All-Chargeability Areas" category. This task required use of the following web site: travel.state.gov/visa/frvi/bulletin/bulletin_ 1770.html.

We specified the web sites and parameters that should be used for the tasks to make it easier to compare usability problems across all of the subjects. Bookmarks were provided to these sites in the test browser to allow subjects to easily access these web sites.

We asked the subjects to think aloud as they worked. The experimenter was available for help if the subjects requested it, but an explicit request was required.

Results

Of the 24 tasks (8 users × 3 tasks), 18 tasks were completed successfully, 4 tasks were not completed, and 2 tasks were skipped due to time restrictions. Inevitably bugs in the software cropped up in some sessions, but the experimenter was able to correct these problems when they occurred and the subjects were able to proceed with the study. A limitation of CoScripter also had to be worked around in all of the studies. CoScripter currently does not correctly pause when a web page makes an Ajax request, so it is necessary for the user to manually insert pause commands into CoScripter scripts that automate Ajax web sites. For this study, the experimenter inserted "wait 6 seconds" commands in three places to assist subjects during the study.

Of the four task failures, two occurred in the first task. In general, subjects struggled with the first task as it was the first time they had actually used the system and Vegemite has a definite learning curve. Subjects generally became more comfortable with the system as they proceeded to the later tasks.

One subject in particular had difficulty with the tool, was only able to attempt two of the tasks, and failed on both. Despite the difficulties, this subject felt confident that he could use Vegemite at the end of the session. This subject was one of the two non-programmers in our study. Of the remaining two task failures, one happened to the other non-programmer subject and the other happened to a programmer. This may indicate that more work is needed to improve our interface for non-programmers.

User Study Discussion

Several users found the sequence of steps needed to construct a mashup overly complicated. One user stated "If I had been given the tool without any instruction, I could not have figured out how to use it. It needs to be more 'discoverable.'" Another user said that it was confusing to use one technique to create the initial table, and another technique to add information to a new column. Note that in the version of the tool that was tested, the bulk data extraction process was completely separated from the script-based nature of CoScripter. We have since added the ability to record the data extraction process as a script, and now the interfaces for extracting initial data and adding new data to the table are much more consistent.

Some of the difficulties were due to user interface details that we can improve in future versions of Vegemite. For example, the creation of a script to get new data for a column and the creation of a new column in the VegeTable are not linked in the user interface. Scripts are also not explicitly linked with the column or columns that they populate. One possible improvement is to add a "New script…" button at the top of every empty column and then replace it with a "Run script" button once the script has been recorded.

Another problem arose because of the separation between Vegemite's primary interface in the VegeTable editor, and CoScripter's primary interface in its sidebar. For example, some users initially tried to click the "Run" button in the CoScripter sidebar instead of selecting the script from Vegemite's "Run" menu. To correct this particular example, we have since added a "Run All Rows" button to the CoScripter toolbar. More work is needed to integrate the CoScripter and Vegemite interfaces, however.

Despite these problems, at the end of their sessions all of the users reported that they were comfortable with the tool, and they felt it would be easy for them to use it to create additional mashups. A few subjects asked if they could get a copy of the tool to use on their own computer.

DISCUSSION AND FUTURE WORK

In Vegemite, the primary method of mashing up data is to take one or more values from a row of a VegeTable, use it as input to perform an action on another web site, and then put the results of that action back into the table. We believe the advantage of this approach is that it matches up more easily with how users would think about extracting this data. In addition, it is easy for the user to inspect how the value of a particular cell was created and modify the value or the script on the fly if a problem is discovered.

This means that Vegemite can use the web itself as a computational fabric. For example, Vegemite does not have built-in currency conversion, but there are numerous web sites that can perform this function. To convert currency within a table, a script simply copies the value from the table into the web site's input form, and then copies the value from the result page into a new cell in the table. Even more mundane functions can be executed using existing web sites, such as time/date arithmetic and string manipulation.

Data transformations are only one benefit of Vegemite's general-purpose ability to mix data from its table with the web. An area that we have not yet explored is using scripts to automate visualization of the data in a VegeTable. For example, a Vegemite script might automate a visualization site such as Many Eyes [20] to allow users to look at their data in new ways.

An advantage of using CoScripter as our scripting engine is its support for mixed-initiative interaction, where users can perform some steps manually if they wish. Mixed-initiative execution saves the user from having to construct code that can handle all corner cases within a data set. If a script does not perform correctly for a particular row, the user can select that row and step through the script, skipping steps that do not work and performing actions manually if necessary. This eliminates much of the debugging overhead caused by corner cases, which is essential since data models from web sites are unlikely to be perfect. Furthermore, mixed-initiative interaction allows the human to handle actions that are difficult to specify computationally (e.g., selecting something based on semantic meaning) and let the system handle the rest.

There are some limitations to the Vegemite approach. Performance is an obvious issue. A script of web page actions is often slow, since there are usually multiple round trips to the web server. These delays are magnified when the script is run for each row to fill a column in a VegeTable. In practice, we have not found this to be a large problem because there are often a relatively small number of rows (around 30 maximum) in the mashups that we have created so far. Users can also leave the process to run in the background while they plan their next steps or work in another application. Furthermore, running a Vegemite script is still faster than performing the equivalent task manually.

It may be possible to address some of the performance issues by taking advantage of the knowledge that a script will be executed multiple times for different rows. Nearly all scripts start with going to a specific URL and occasionally some additional navigation steps, which will always be executed in the same way every time the script is run. Vegemite could use an approach similar to Smart Bookmarks [8] to save the state of the page after the common page navigation script steps have been executed, and then subsequent runs of the script could start from the saved page state rather than at the beginning. Alternately, after a script has run it might be possible to use the browser's back button to step back to the point where the next script's execution will diverge. We have not experimented with any of these ideas however, and implementing performance optimizations is the subject of future work.

Another issue for Vegemite is ambiguity in the data that should be collected from a given page. Currently, our primary means of copying data is through the "copy text following a label" rule, which has worked reasonably in practice but will fail if Vegemite cannot identify a label for the extracted data or the label that was found changes across script runs. We plan to address this limitation through two approaches. First, we are interested in applying generalization techniques found in programming-by-demonstration systems such as Eager [2]. Second, we plan to extend Vegemite so that it can understand the semantics of the page it is acting on, through microformats [15] or data detectors [6]. For example, if a user copies a person's name from a list of people that was marked up with the hCard microformat, Vegemite could record the action as copying a person's name, as opposed to just copying text. This would improve the robustness of the scripts that Vegemite can record and execute.

One important type of ambiguity that may be harder to address occurs when Vegemite must parse search results returned by a web site and choose the intended result. For example, when searching for a film in a public library, the search results may include a novel of the same name, a videotape version, and a DVD version. If the user's goal in creating the mashup is to find out if their local library has a DVD of a film, the user must be able to specify how to select the record of interest. Currently, Vegemite supports only order-based selection of values (e.g., the third result). More work is needed to explore and understand how the user could specify, for example, that they are only interested in "DVD" results.

Data on the web is often in the wrong format or has errors. Vegemite allows users to directly edit the data table to correct minor mistakes; however, it does not currently support automatic or semi-automatic data cleaning. One way to address this is to adopt an approach similar to Karma, where the user cleans one data item in a table, and Karma infers the data cleaning transformation and applies it to the other data in that column. Another approach is to use a system like Topes [17] for end users to specify and apply data cleaning transformations.

More work is also needed to study and improve on the usability of Vegemite, especially for non-programmers. We saw from the user study that the Vegemite tool has a definite learning curve, and that our programmer subjects were much more comfortable using the system after having used it to create several mashups. Non-programmers may have even more difficulty. In the future, we plan to do a more comprehensive user study involving only non-programmers to better understand the issues that these particular users face.

Another area of future work is to explore where the Vegemite tool should be located within the users' environment. A point of contention among the authors is whether Vegemite should

be integrated more tightly with CoScripter and the web browser, or whether Vegemite might be more useful if it were integrated with an existing spreadsheet tool. In the spreadsheet case, CoScripter scripts would become custom formulas that users could apply to other data within their spreadsheet. Such an idea would be somewhat reminiscent of the A1 system for helping system administrators [13], though a key challenge would be maintaining the mixed-initiative style that is so powerful in the current design.

CONCLUSION

Vegemite is distinguished from other mashup tools by 1) its focus on programming by demonstration; 2) iterative, interactive transformation of data by the user; and 3) mixed-initiative interaction. These features lower the barrier to creating mashups and relax the assumption that the program must perfectly handle all cases or not handle them at all. By making mashups easier to build, more users can construct mashups on the fly as needed to meet their immediate, idiosyncratic needs.

ACKNOWLEDGMENTS

We thank our user study participants for helping us evaluate Vegemite, and the rest of the CoScripter team for their valuable comments.

REFERENCES

1. Bigham, J.P., A.C. Cavender, R.S. Kaminsky, C.M. Prince, and T.S. Robison. Transcendence: enabling a personal view of the deep web In *Proc. IUI 2008*. ACM Press (2008), 169-178.
2. Cypher, A., Eager: Programming Repetitive Tasks by Demonstration, in *Watch What I Do: Programming by Demonstration*, A. Cypher, Editor. MIT Press. pp. 205-217, 1993.
3. Dontcheva, M., S.M. Drucker, D. Salesin, and M.F. Cohen. Relations, Cards, and Search Templates: User-Guided Web Data Integration and Layout. In *Proc. UIST 2007*. ACM Press (2007), 61-70.
4. Dontcheva, M., S.M. Drucker, G. Wade, D. Salesin, and M.F. Cohen. Summarizing Personal Web Browsing Sessions. In *Proc. UIST 2006*. ACM Press (2006), 115-124.
5. Ennals, R. and D. Gay. User-Friendly Functional Programming for Web Mashups. In *Proc. ICFP 2007*. ACM Press (2007), 223-234.
6. Faaborg, A. and H. Lieberman. A Goal-Oriented Web Browser. In *Proc. CHI 2006*. ACM Press (2006), 751-760.
7. Fujima, J., A. Lunzer, K. Hornbaek, and Y. Tanaka. Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access. In *Proc. UIST 2004*. ACM Press (2004), 175-184.
8. Hupp, D. and R.C. Miller. Smart Bookmarks: Automatic Retroactive Macro Recording on the Web. In *Proc. UIST 2007*. ACM Press (2007), 81-90.
9. Huynh, D., S. Mazzocchi, and D. Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In *Proc. ISWC 2005*. Springer (2005), 413-430.
10. Huynh, D., R.C. Miller, and D. Karger. Enabling Web Browsers to Augment Web Sites' Filtering and Sorting Functionalities. In *Proc. UIST 2006*. ACM Press (2006), 125-134.
11. Huynh, D., R.C. Miller, and D. Karger. Potluck: Data Mash-Up Tool for Casual Users. In *Proc. ISWC 2007*. Springer (2007), 239-252.
12. IBM, IBM Lotus Mashups. http://www.ibm.com/software/lotus/products/mashups/
13. Kandogan, E., E. Haber, R. Barrett, A. Cypher, P. Maglio, and H. Zhao. A1: End-User Programming for Web-based System Administration. In *Proc. UIST 2005*. ACM Press (2005), 211-220.
14. 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 *Proc. CHI 2007*. ACM Press (2007), 943-946.
15. Microformats. http://microformats.org/
16. Microsoft, Microsoft Popfly. http://www.popfly.com/
17. Scaffidi, C., B. Myers, and M. Shaw. Topes: Reusable Abstractions for Validating Data. In *Proc. ICSE 2008*. ACM Press (2008), 1-10.
18. Solvent. http://simile.mit.edu/wiki/Solvent
19. Tuchinda, R., P. Szekely, and C.A. Knoblock. Building Data Integration Queries by Demonstration. In *Proc. IUI 2008*. ACM Press (2008), 170-179.
20. Viégas, F.B., M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon, Many Eyes: A Site for Visualization at Internet Scale. *IEEE Transactions on Visualization and Computer Graphics*, 2007. **13**(6): 1121-1128.
21. Wong, J. and J. Hong. Making mashups with Marmite: towards end-user programming for the web. In *Proc. CHI 2007*. ACM Press (2007), 1435-1444.
22. Yahoo!, Pipes. http://pipes.yahoo.com/