Mining Web Interactions to Automatically Create Mash-Ups

Jeffrey P. Bigham
Dept. of Computer Science
University of Rochester
Rochester, NY 14627 USA
jbigham@cs.rochester.edu

Ryan S. Kaminsky
Dept. of Computer Science & Eng.
DUB Group
University of Washington
Seattle, WA 98195 USA
rkamin@cs.washington.edu

Jeffrey Nichols
USER Group
IBM Almaden Research Center
650 Harry Rd
San Jose, CA 95120 USA
jwnichols@us.ibm.com

ABSTRACT

The deep web contains an order of magnitude more information than the surface web, but that information is hidden behind the web forms of a large number of web sites. Meta-search engines can help users explore this information by aggregating results from multiple resources, but previously these could only be created and maintained by programmers. In this paper, we explore the automatic creation of *meta-search mash-ups* by mining the web interactions of multiple web users to find relations between query forms on different web sites. We also present an implemented system called TX2 that uses those connections to search multiple deep web resources simultaneously and integrate the results in context in a single results page. TX2 illustrates the promise of constructing mash-ups automatically and the potential of mining web interactions to explore deep web resources.

ACM Classification H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General Terms Design, Human Factors, Algorithms

Keywords: Mash-Ups, Programming-by-Example, Meta-Search, Deep Web, Web Forms

INTRODUCTION

The deep web contains an incredible amount of information that is only accessible by querying web forms, making it difficult for web crawlers to automatically access. As a result, users are usually restricted to accessing one resource at a time instead of aggregating information from multiple web sites simultaneously, and must use the interfaces provided by each individual site. Meta-search engines aggregate results across specific sites, but must be created and maintained by programmers. Prior approaches to the difficult problem of crawling the deep web and creating meta-search engines automatically have primarily constructed relations by examining the low-level HTML form schemas within web pages. This paper introduces TX2, a browser extension that finds relations between query forms by mining the interaction history of many web users and also allows users to explicitly map query forms through interactive demonstration.

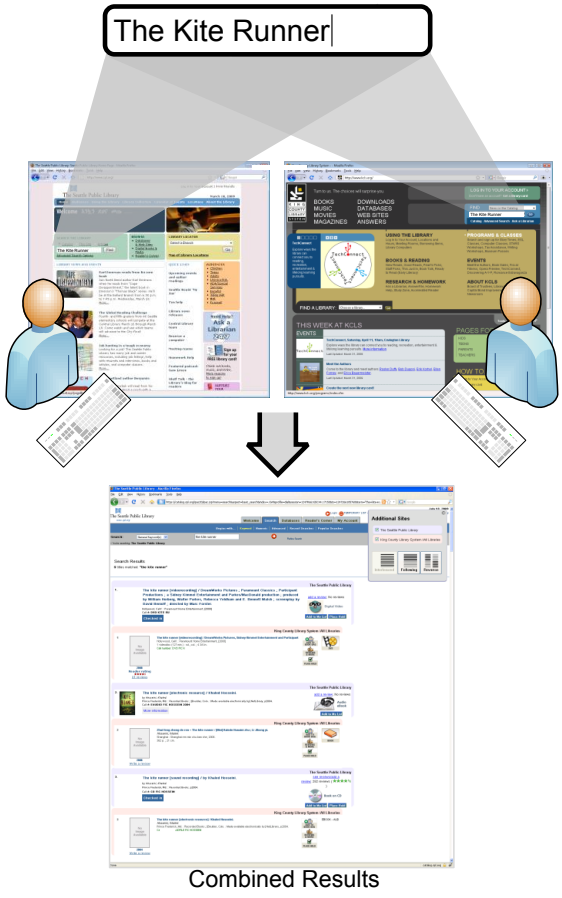

Figure 1: The web usage of multiple web users often implicitly connects the forms of related web resources. In this example, two users have input the same query ("The Kite Runner") into different search forms: one on the Seattle Public Library and the other on the King County Public Library. TX2 can automatically make available results from both resources in response to a query on either.

A *meta-search mash-up* is a mash-up that aggregates the search results of form queries to multiple deep web resources. TX2 makes possible the automatic creation of meta-search mash-

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

ups by mining the observed interactions of its users. Prior work in this area has required people to explicitly create mash-ups, most often programmers using programming tools. Some tools are designed for end users, such as d.mix [9], Yahoo Pipes [24], Clip, Connect, Clone [8], and Vegemite [12], which provide interfaces that support the creation of certain mash-ups without low-level programming. Creating mash-ups automatically has the potential to bring the benefits of mash-up creation to not only technically-savvy end users, who are beginning to have access with recent tools, but also to anyone who can use the web. To the best of our knowledge, TX2 is the first system to use recorded interactions to automatically connect deep web resources.

To illustrate TX2's motivation, consider the following example illustrated in Figure 1. In the Seattle area, there are multiple libraries available at which the public can check out books, including the University of Washington Library, the Seattle Public Library, and the King County Library. Any single library might not yet have a particular hot new release or obscure title, but collectively the chances are much higher. To find their desired book users might need to search the web sites of all three libraries. By entering the same value into the forms on multiple sites, users implicitly connect the forms on each of these resources. TX2 leverages recordings of interactions such as these to create custom meta-search mash-ups that present users with results from all three resources.

Importantly, a single user does not necessarily need to enter the same information into each of the different forms. If web interactions from multiple users are shared, TX2 can facilitate the discovery of new resources. Alternatively, users who want to create new meta-search mash-ups could create them explicitly by submitting the same search queries on each of the three sites.

TX2's motivation includes the following:

- **Creating mash-ups and meta-search engines is costly:** Because TX2 creates mash-ups automatically, there is no overhead to getting started using TX2. Prior meta-search mash-ups created for exploring the deep web required programmers to explicitly define the meta-search capabilities of each site and manually specify how they should be connected. This meant that meta-search was primarily only available for popular sites and sites at the long tail lacked meta-search capabilities.
- **The best sites often change:** The best sites for providing a specific kind of information often change, but many users remain unaware. For instance, in the past fifteen years, the list of the best search engines has changed multiple times[1]. Among travel sites the changes have been even more dramatic. A few years ago, the airlines realized that they would prefer customers to shop directly from them instead of through sites like Travelocity and Expedia, and began guaranteeing that the lowest prices would be found on the airline sites themselves. This led to the prominence of meta-search sites like *kayak.com*, which today most often lists the lowest prices. TX2 allows the easy discovery of such a transition. As more people switch to using a new site for a particular kind of information, TX2 will begin offering results from those new sites to users of the old sites automatically.
- **People want great interfaces AND great information:** Some web site interfaces work better for people with specific needs. For example, some sites may work better on small screens or may be more accessible to blind users (highly-dynamic sites are often difficult for screen readers to process). TX2 lets users independently choose the interface and the sources of their information.

[1]The authors note sadly the fall of hotbot.com

TX2 makes the following three contributions:

- An algorithm for mining meta-search mash-ups from the web usage logs of one or many web users, and a study using real data demonstrating that this approach can work in practice.
- A general method for automatically identifying and combining the results from queries to multiple web sites, allowing client-side meta-search mash-ups.
- A general illustration of the promise of using recorded interactions for both automatically creating meta-search mash-ups and for exploring deep web resources.

In the remainder of this paper, we will first discuss related work. Then we will present a study showing that users are already implicitly defining meta-search mash-ups and describe our new techniques for extracting these relationships. Next we will discuss a tool which brings these new mash-ups to users with a browser extension that records user interactions, automatically discovers similar deep web resources across different sites based on the recorded user interactions, issues queries to multiple similar resources simultaneously, and then presents them to the user in place on the site with which they are already familiar.

RELATED WORK

Work related to TX2 falls into two general areas: existing tools for exploring deep web resources, and work that helps users collect, manipulate, and mash-up web content.

Exploring the Deep Web

The deep web is that part of web content that is accessible only after submitting a web form. Existing tools for exploring the deep web resemble traditional, surface web crawlers [14], and have focused on the following two tasks: (i) identifying deep web resources that should be connected, and (ii) matching the web forms of multiple resources together. TX2 leverages user interactions to accomplish these difficult tasks.

Deep Web Crawlers HiWE [19] and MetaQuerier [4] seek to automatically identify both web forms of interest and the data types of their fields. This is accomplished by analyzing the content of web pages, including both the human and machine readable labels associated with forms and their fields. When successful, these systems can connect the web forms of related resources together. TX2 uses recorded user interactions, in addition to page content, to find both related resources and the connnections between their input fields.

Connecting deep web resources requires both finding the related deep web resources and matching the schemas of web forms to known schemas automatically [5]. This is difficult

for a number of reasons, including that form fields are not always assigned meaningful labels and many labels are repeated (for instance, "Keywords:"). The web interactions of users reveal both types of information. To the best of our knowledge, TX2 is the first system to mine user interactions to link deep web resources together.

User interactions have been shown to reveal much about web resources. For example, Adar *et al.* identified a number of distinct patterns in web site usage logs and associated them with higher-level semantics of the sites [1]. TX2 looks at web usage at a lower level, using observed interactions with web page components to infer semantics of how they are used, and which components on other web sites might be related.

Meta-Search Engines Some of the earliest (and most popular) tools designed to explore the deep web were meta-search engines. Meta-search engines let users search multiple deep web resources simultaneously. One of the earliest examples was MetaCrawler [20], which aggregated and reranked the results from multiple search engines available at the time. Shopping meta-search sites followed soon after, aggregating product information from many shopping sites to help users find the best deal. Examples include *kayak.com*, which aggregates the results of multiple airline sites, and MySimon and Froogle, which aggregate pricing information from multiple shopping sites. Many libraries include search options across other libraries, although interestingly, these relationships often only loosely match to geographic locality. (The Seattle Public Library and King County Library do not provide an option to let users search one another's holdings even though Seattle is located in King County.) Numerous sites that one might like to be able to search together do not have meta-search provided for them.

Tools for End Users Several end user tools help users explore deep web resources, but have focused on helping users explore one deep web resource at a time. Sifter detects and augments the interfaces for existing result pages on a single deep web resource [10]. Transcendence helps users issue multiple queries to a single deep web resource in order to facilitate previously unsupported queries [2]. Neither lets users aggregate the information from multiple resources simultaneously. To the best of our knowledge, TX2 is the first tool that automatically aggregates information from multiple deep web resources.

Web Mash-Ups

Web mash-ups combine information from multiple sources. Existing systems enable both programmers and end users to flexibly manipulate, store and merge web data information. User scripting systems, such as Greasemonkey [18] and Chickenfoot [3], modify web pages according to user specifications. Piggy-Bank users can write scripts to extract information from web pages and store that information in a semantic web database [17] and Solvent facilitates the creation of Piggy-Bank scripts with visual tools [21].

Other systems extract information from multiple web pages and integrate results together on a single page. Web Summaries [6] can explore multiple links selected by users but does not address form submission, required for exploring the deep web. The cloning feature of Clip, Clone, Connect (CCC) [8] lets users submit multiple form queries, but these mash-ups must be explicitly created and results are not added in the context of an existing web site. TX2 makes the process of creating mash-ups transparent to users; from the user perspective, new results simply appear along with the results they would have seen otherwise.

Many systems let end users create mash-ups. Marmite [23] and Vegemite [12] help users select web data and then manipulate it in a familiar interface similar to a spreadsheet. Yahoo! Pipes [24] uses the visual metaphor of a pipe in a program that enables users to connect web resources . d.mix [9] copies parametric versions of mashed-up web elements into a spreadsheet interface for manipulation by users. Reform [22] lets programmers and end users work together - programmers define mash-ups that can be applied to many different sites by end users . End users can apply mash-ups with relative ease, but each new type of mash-up requires programming. TX2 automatically creates broadly-applicable meta-search mash-ups based on user interaction without programming. Although these interfaces make the creation of mash-ups more accessible to end users, they require end users to expend effort in creating mash-ups and understand the underlying concept of how mash-ups are created. Because TX2 has the potential to create mash-ups without explicit user interactions, even novice users could benefit from it.

Web macro recorders, such as Creo [7], PLOW [11], CoScripter [13] and Turquoise [15], could be used to collect results from a single web site but cannot merge the results from multiple sites. In TX2, users define forms that they want to explore by searching with those forms, but they do not need to be aware of an explicit recording phase.

SYSTEM DESCRIPTION

TX2 consists of the following three logical components:

- **Recorder** - records user interactions using technology borrowed from CoScripter [13].
- **Connector** - collects user interactions and provides a service that returns a set of forms from other web sites that are similar based on interactions on the current web site.
- **Integrator** - collects results from multiple related resources and combines them on a single existing results page on-the-fly, allowing users to control how and when they are displayed.

All three components are currently implemented within an extension to the Mozilla Firefox web browser. Eventually, we plan to deploy the connector as a web service where data will be incrementally added by TX2 users to a shared repository (Figure 2). In anticipation of this, the connector algorithms and data structures are designed to work online and incrementally. For testing purposes, we added an existing history of interactions from multiple users to the connector.

To better illustrate how the system works, we will explain its functionality in the context of the following scenario: TX2 user Vella is searching for a flight on the United Airlines web site. At a high-level, Vella will visit *united.com*, enter some values into various fields of the reservations form, and click the "Submit" button. At this point, her browser

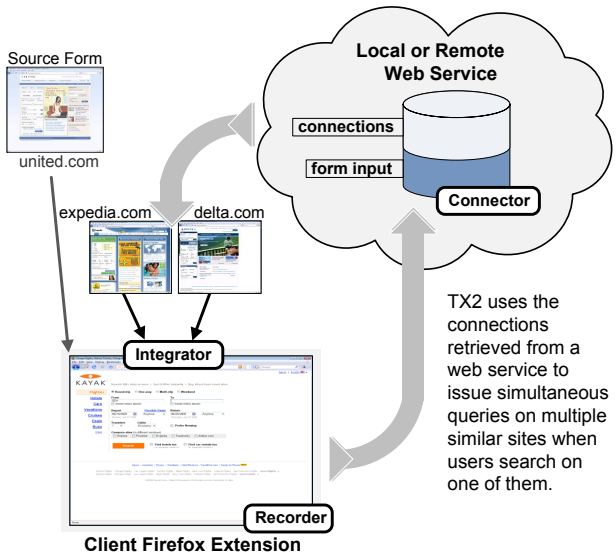

Figure 2: The TX2 Firefox extension (i) retrieves connections between different deep web resources from the TX2 web service that has mined them from user interactions, and (ii) applies these connections as users browse the web to automatically create meta-search mash-ups.

will perform the normal HTTP request with *united.com*, but simultaneously TX2 will contact the connector, retrieve a list of matching forms, submit those forms with the data Vella entered on *united.com*, and collect new results from the resulting submission. When the results page for *united.com* appears, TX2 will integrate results from the other sites into the results listed on *united.com*. The low-level details of how the system works will now be discussed.

The Recorder

TX2's recorder is built on top of the CoScripter platform for recording and playing back user interactions with web pages [13] and uses CoScripter's ActionShot extension for continuously recording users' web activity. Unlike some other web recording frameworks that record web history based purely on page views and URLs, CoScripter records low-level interactions, such as clicking on a button or entering text into a textbox. CoScripter also represents these events in a pseudo-natural language format known as "slop" which includes a string-based label identifying the target of an interaction and also a string-based value to enter into that target where applicable. The Connector's data mining algorithms make use of the information within these "sloppy" descriptions, along with other contextual information, to identify potentially matching query forms on different web sites.

In the context of our scenario, the recorder is always turned on and recording Vella's interactions with her browser. When she visits *united.com*, the system records that she typed the URL "united.com" into the browser window, and then entered the airport codes "SEA" and "BOS" into the "From" and "To" textboxes, respectively. It then records that she entered her departure and return dates into the appropriate text boxes. When Vella clicks the "Submit" button, the system bundles together all of the actions recorded following the entry of the URL, because that step is the most recent step that led to a new web page. The bundled steps are then sent to the connector, along with meta-data about each of the steps (such as XPaths of the targets). This information is used to find other web sites with forms similar the one Vella completed on *united.com*.

The continuous background recording functionality needed in this scenario was added to CoScripter by ActionShot, which also provides a user interface for interactively browsing and searching through this history. Currently, ActionShot only allows users to explicitly share portions of their web history with friends through social networking sites and e-mail. It does not automatically send recorded information to a central server. In the future we may add this functionality, but will carefully consider the privacy and security concerns of deploying such a system before doing so. Some of these issues are discussed later in the "Discussion" section.

For this paper, we asked six regular users of the ActionShot prototype to donate three months of the browsing history ActionShot had collected for them. This data was primarily collected on work machines, however two of the participants recorded data on their primary machine (both laptops). We did not clean or anonymize this data in any way after collection, however the users were well aware of the eventual use for this data and were allowed to delete any sequences of actions from their data that they did not want to share. The resulting data set contains 13,159 interactive steps across 5459 web pages. Multiple interactive steps were taken by participants on 1796 of those pages. Most of these presumably involve entering data into forms because the only recorded interactive steps that do not involve forms are clicking on links, which usually cause a new page to load.

The Connector: Mining Recorded Interaction Histories

The connector takes as input a sequence of interactive steps that a TX2 user has performed on a query form (the source form) and meta-data for each step, including the XPath for each form field. The connector's output is a ranked list of potentially matching forms (the target forms). For each target form, a mapping is given between the XPaths of the fields on the source form and the XPath of the fields on the target form. The connector can also return a sequence of instructions in CoScripter slop to execute on each of the target forms, however TX2 currently only makes use of the XPath mapping.

The connector performs two functions: data extraction and aggregation, and matching over the aggregated data.

Data Extraction and Aggregation The data extraction and aggregation component operates online as new matching requests are received, generating and updating a set of basic data structures to facilitate matching requests. The data extraction and aggregation process is comprised of the following steps:

1. **Filter individual actions and generate page sets.** Recorded actions from users are added to the repository as a stream. Normally, user interaction data submitted to the system would be from a single page, however the data collected by our test users spanned many pages and sites. In order to segment the test data, the stream of actions were

Name	Type	Description
Label	String	A human-readable label identifying the action's target element
Value	String	The value entered into the target element
Date	Boolean	True if the value entered contains a date
Airport Code	Boolean	True if the value entered contains an airport code
Airport Full Name	Boolean	True if the value entered contains a standard airport full name
Zip Code	Boolean	True if the value entered contains a zip code
Page Title	String Array	An array of all words in the page title that are five characters or longer
Label Keyword	String Array	An array of all words in the label that are five characters or longer

Table 1: Feature generators implemented as part of the TX2 data matching component.

first split into sets of contiguous actions that occurred on a single page URL. Each of these sets is called a "page set." Any page sets containing only a single action were thrown out, as we assume a form interaction must involve at least two form elements (a textbox and a submit button, for instance).

When Vella requested matches, the five interactions that she sent were bundled into a new page set by the system.

2. **Feature generation for actions and page sets.**
TX2 uses "feature generators" to extract potentially meaningful information about actions. For example, the label generator attempts to extract a human-readable string label from each action. If a label can be found, the label value is associated with the action. We have implemented several feature generators for semantic types, which are described in more detail in Table 1.

   All of the actions that Vella completed on *united.com* involved elements with associated labels - the "From", "To", "Departing" and "Returning" textboxes and the button labeled "Search." In addition, Vella entered strings of type "airport code" into two textboxes and strings of type "date" into two others. Our feature generators can detect these semantic types and adds the appropriate features to the actions contained within Vella's page set.

   In order to facilitate matching, features are aggregated by data structures that contain actions, such as the script models and meta-steps discussed in the next section.

3. **Script model generation**
At this point, we have a long list of many page sets, some of which may describe different interactions on the same page. We start by grouping page sets by the URL that they operate upon, and from these groups we generate an abstract model of a script that acts on a page. We make three assumptions when generating this model: (i) there is only a single query form on this page and thus all actions in all page sets must be operating on the same form, (ii) actions in different page sets that target the same label must also target the same element, and (iii) the order in which actions occur on the page (except for clicking the final submit button) is not meaningful.

   The first assumption is necessary because ActionShot does not currently provide sufficient information to distinguish between operations on different forms on the same page, although this could be obviated if more information was recorded. The second assumption is reasonable because CoScripter's labeler generates a unique label for each element. The third assumption is reasonable for most web forms, although examples exist that violate it. For example, some pages update the options available in one combo box based on a selection in another combo box. It is likely possible to use ordering information in the example scripts to infer ordering dependencies among actions, however we leave this for future work. These assumptions all appear reasonable in practice, based on our repository.

   A script model is generated for a given URL by iterating over all of the actions in the page sets that operate on that URL. Actions with the same label are grouped together as a "meta-step." Each meta-step then represents a step that might be taken in the script model and aggregates within itself multiple example actions. For instance, the meta-step recorded for Vella's interaction with the "From" textbox on *united.com* would contain not only the value "SEA" that she entered along with a true value for the "airport code" feature, but also the values entered by other visitors to the site, for example ("CMH", true) and ("Tacoma", false). When this process ends, each script model is populated with a set of meta-steps.

The end result of the data extraction and aggregation process are the script models which facilitate matching. These algorithms can be executed incrementally by generating new features and updating existing script models with new action data as it is acquired.

Matching Aggregated Data The matching algorithm is executed when a query for matching forms is received, such as when Vella submitted her form on *united.com*. This query contains the set of actions performed on the source query form in the CoScripter sloppy format. The query actions are first mapped to a script model in the repository using the source query form's URL. If a script model already exists, then the query actions are added to the model incrementally using the process discussed above. If no script model exists for the form, then a new model is created from the query actions. Features are also generated for all of the new actions using the method discussed above. Note that the act of querying our service has the side benefit of updating our knowledge base.

Once the query has been associated with a script model, the connector attempts to identify related forms with a two phase process: first, it attempts to identify other script models that may be related, and then attempts to match the meta-steps on the source script model with those in the target script model.

Identifying potential matches at both the script model and meta-step levels is performed using the same basic process. First, features are aggregated from all constituent actions in the objects being compared. Second, features are compared and match counts are generated reflecting the number of features that were shared by the object. Features are weighted differently when generating the match count. For example, semantic features such as airport or zip codes are weighted more highly than labels or values. Actions can be found to match one another based on the labels assigned to their target elements, values provided as input, or the inferred semantic types. Features change infrequently and match counts can be cached for improved performance.

The first matching step is to identify additional script models that may be potential matches for the source script model. A list of possible models ranked by match count is generated using the matching process described above. In our current system, any model with a match count greater than zero is included in the list, which seems to be reasonable given the size of our data set. As the amount of data in the system increases, it would likely be necessary to set a higher threshold.

For each potential target script model, we then want to match meta-steps in the source model to meta-steps in the target model. It is possible that not every source meta-step is relevant to our query. Many forms contain optional fields and not all meta-steps may have corresponding steps in a query. In our implementation, we make an assumption that only the actions mentioned in the query are relevant and must be matched. Other meta-steps in the source model that do not have corresponding actions in the query are ignored for the remainder of this process.

In order to determine a match between the meta-steps in the source and target script models, we construct a "pair list," which contains every possible pair of source meta-steps and target meta-steps, and generate match counts for each pair in the "pair list." We also construct an empty "match list" to hold matching pairs. We assume that only one source meta-step can match one target meta-step (and vice versa). We then find the pair in the "pair list" with the highest match count, remove it from the "pair list," and add it to the "match list." We then remove all other pairs from the "pair list" that contain either of the meta-steps in the matched pair. This loop is performed until the "match list" contains a pair for every step in the source query or until all pairs in the "pair list" have match counts of zero. If the former condition is true, then we have found a matching form. From the pairs in the "match list," we can return either CoScripter slop corresponding to the target form or a set of XPath mappings between the source and target form.

The matches returned to the TX2 client may still not all be correct. For example, a matched target form may require more values than were specified in the source query. The TX2 client has enough information to attempt to submit the target form, but in this case the form will fail. This is typically easy for TX2 to detect and such failures can often be ignored as we might assume a form failure indicates that the target form was not an adequate match for the source query.

The Integrator: Facilitating Meta-Search on the Client

The main function of the integrator is to merge the existing results of a web query with the results from additional, related web resources returned by the connector. TX2 uses multiple crawling threads simultaneously to quickly retrieve the results from connected sites to form the meta-search mashup. These crawling threads use full browser windows (hidden from the user) that both fully render the retrieved content and execute any scripts contained on the downloaded pages.

TX2 first detects form input on each page that is loaded, uses the connector to find matching forms, and finally submits input to each matching form using one of several crawler threads running in the background. The main difficulty here is handling the wide diversity in how forms are created and input is sent to web servers. The crawlers used by TX2 fully load the web page containing each form that has matched. It then fills out each field of the form, calls any scripts that need to be called prior to submitting, and finally submits the form.

Because the TX2 crawlers go through the entire web transaction - from loading a form, to submitting it, to viewing the results - TX2 is robust to a variety of different web forms (although not all, as will be discussed later in the "Evalution" section). Forms can use HTTP GET or POST, submit via programmatic calls, or even set special variables using scripts before the form is submitted. A server-side script will observe little or no difference between TX2 and a live person filling out and submitting a supported form.

Detecting and Integrating Results The TX2 crawlers assume that a list of results will be returned after a form is submitted. Results are, for instance, the search results returned by Google, the flight options returned to Vella by *united.com*, or the products returned by Amazon. TX2 uses the repeating DOM structures characteristic of submission results to automatically select individual result fields.

To find an initial list of likely parent nodes, TX2 uses a heuristic similar to that used by Sifter [10]. In this iterative process, TX2 starts with an initial list of links and during each round adds a "../" to the current XPATH, proceeding until a smaller number of nodes is selected. This process helps find the most general result that maximizes visual layout while preserving the cardinality of the set of matching records.

TX2 next selects a number of fields from within the results of the crawler browsers that are either links or identified semantic types (currency, numbers, or names). It then identifies which of these fields are also present in a majority of the other records. Results not containing at least 80% of these popular fields are pruned. This two step process includes as many results as possible without adding content that is unlikely to be results.

TX2 preprocesses each document that it loads to improve selection accuracy [2]. First, TX2 adds additional structure to pages that express results as sequential patterns of elements, instead of as separate DOM elements. Sequential patterned elements are often difficult for DOM-based extraction systems to interpret correctly [10]. To address this problem, TX2 uses a recursive process inspired by Mukherjee *et al.*

[16] to add additional elements to the DOM to accurately express records characterized by repeating elements. A second preprocessing step promotes text nodes containing common semantic types to element nodes to ensure that they can be chosen as fields.

Content on many web pages changes after the page is loaded. TX2 crawlers can integrate results from dynamic, AJAX-driven deep web resources. As an example, a flight search on $expedia.com$ first returns in a page that asks users to wait for their results. Only later is the page populated with flights that have been found. It is intractable to determine when a dynamic page has finished updating in a general way, but TX2 uses a heuristic that lets it incorporate new results without making the user wait for results that may never come.

If, after a form submission, no results are detected, each crawler thread waits a short amount of time (currently 30 seconds), periodically checking for new results. If no page changes occur and no new pages load during this time, TX2 assumes that no new results will be added. Because multiple crawling threads run in parallel, the user does not have to wait on results from this process to receive results from the other sites that are being searched in the background.

MATCHING RESULT STRUCTURE & VISUAL STYLING

TX2 can present results in two ways: (i) preserving the original appearance of each result on the page from which it was taken and (ii) matching the appearance of each result to the visual styling of the page on which it is displayed.

Preserving the Visual Styling of Source Web Site

For clarity of result provenance, TX2 defaults to presenting results in their original formatting. Before each result is imported from its crawler browser, its styling is hard-coded into the element. This step captures and preserves styling inherited from nodes higher in the DOM tree.

The proper visual display of elements may also depend on styling associated only with their parents in the tree. For instance, a result element may use a light-colored font which is only visible because a parent uses a dark-colored background. To facilitate correct display in such situations, TX2 travels back up the DOM from each result, collecting the style information of each parent. It then synthesizes these separate styles into a single style. A new container node is added as a parent to each result and assigned that style.

Matching Visual Styling

To preserve the appearance of results, TX2 can also attempt to match the visual styling of new results to those on the page to which they are added. This makes Google results look like Yahoo results or results from *united.com* look like those from *delta.com*. In many cases, the origin of a result does not directly determine its utility, and, for users accustomed to a particular presentation style, that style might be more preferred or understandable. Users may prefer the information on one site but the interface of another.

Even results that are visually similar may have very different underlying structures, making mapping one result to another non-trivial. The algorithm used to match one result to another is based on the following observations. First, results usually contain some information that is more important than the rest - for instance, a link to additional information or a price is likely to be more important that other text. Second, the structure and order of presentation is often (but not always) consistent across multiple related sites.

TX2 combines visual styling and DOM structure to map each result to the template result. The first step is to construct a template node from the results on the presentation site. To create this node, TX2 individually compares each result node with every other result node, checking to see if each XPATH element from the first result appears in the second. It computes the overlap between each pair of nodes, and sums them together to derive a final score for each node. The node with the highest score is likely to be most similar to the other results, and is in that way the most typical node. This node is used as a template for creating the style of nodes gathered from other sites.

A deep copy of the template node is made for each result extracted from another site, and the extracted node's content is matched to the template node. This is done using a greedy algorithm. Limited semantics are added to both nodes, and matches are considered in the order of links, semantic types (currently consiting of currency, URL, and numbers), and, finally, all text nodes. All text and image content in the extracted node are added to the template node, regardless of whether a good match was found, helping to prevent users from missing out on information when the matches fail. Although the success of this approach varies from site to site, it is reliably able to match links and semantic types, which are likely to be most important, and place this content in the appropriate place in the template node.

Authoring New Interfaces to Existing Sites

TX2 can be used to create new interfaces for existing sites. To create a custom meta-search mash-up, authors first create a new form that asks for the same information as the interface to be replaced and then presents the results in a desirable format. To connect an existing resource to the new one, they just use both forms to search for the same information.

Applications of this model include creating custom versions of web sites that are more accessible or usable for certain populations (for instance, blind or cognitively impaired individuals), or creating completely custom mobile versions of existing resources. For example, if users found that the Atlanta Public Library worked better on their small screens than the Seattle Public Library, they could choose to have library results displayed using the Atlanta interface using a mobile implementation of TX2.

USER INTERFACE

TX2 users do nothing other than what they normally would in order to create and use meta-search mash-ups. Users just search on existing sites and, if the TX2 web service finds a match, it will search those sites with the user's query terms in the background. If multiple sites are searched, TX2 adds an interface to the page that lets users manipulate the new results (Figure 3). Results from each site are interleaved.

The source of each result is shown both by a color-coding and a source title shown above and to the right of each result.

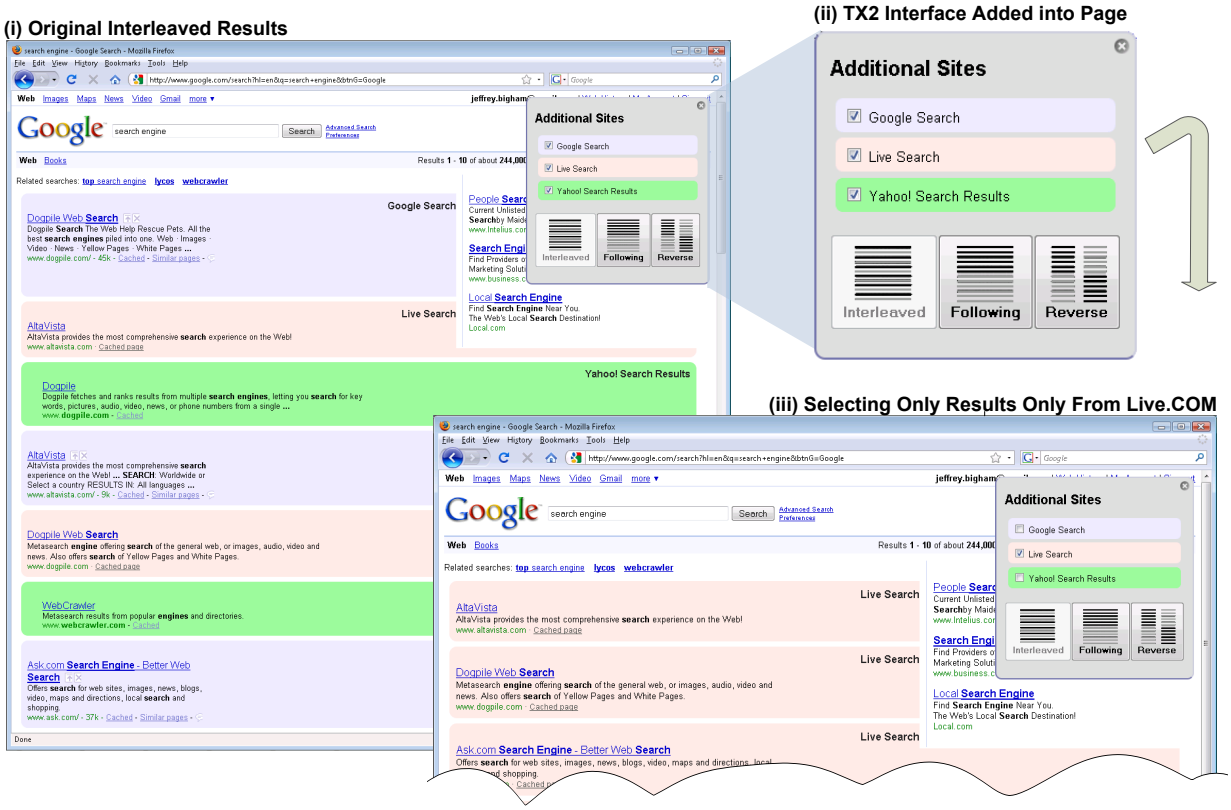

Figure 3: The result of searching Google with the query "search engine" when using TX2. *(i)* Results from Google, Live, and Yahoo Search are all included on the single results page in the context of the Google results page. The source of each is identified by a title and by a color-coding. *(ii)* Users can change the order of the results returned or filter out sources that they do not want using an interface embedded into the page. *(iii)* The users has opted to show only Live results on the Google Results page.

These titles are parsed automatically from each site on the fly using the title of each search results page as a base. If the title contains properly capitalized words, words containing only lowercase letters, or stand-alone punctuation (for instance, -), are removed.

Because TX2 determines the structure of results, users can manipulate them within the context of the host results page using provided controls. Currently, they can reorder results according to provenance, making them appear sequentially, interleaved or in reverse order. They can also choose to show or hide results from particular sources, something which TX2 could use to learn which sources are preferred either collectively or by individual users.

EVALUATION

To determine the feasibility of our approach, we conducted evaluations of both the data mining matching algorithms and the content matching algorithms.

Data Mining Algorithms

To understand the behavior and accuracy of our matching system, we first evaluated it on manually-created matches. We created a data set for testing by manually recording the process of filling in seven different query forms on three different categories of web sites. For the categories, we chose Libraries, Travel Sites, and General Search sites. Within the Library category, we recorded queries at the US Library of Congress web site and the Seattle Public Library. Within the Travel category, we recorded queries at *kayak.com*, *orbitz.com*, and *travelocity.com*. For General Search we recorded queries of Google and Yahoo. We added this manually-recorded data to our existing corpus, so that we could test both our performance on the target sites and see whether any other sites from our corpus matched the manually-recorded sites.

TX2 was able to match the manually-recorded sites in each category with one another, and also find matches in the existing corpus. For example, *southwest.com* was found as a match for the various travel sites, the Santa Clara Library web site was found as a match for the other two library web sites, and the generic search engines also matched with many of Google's other specialized search engines, such as Google News and Google Image Search. In a few instances, unwanted matches were also found. For example, the real estate web site *redfin.com* was common in our corpus and was matched with the generic search sites.

We also found some problems with the matching algorithms which will need to be considered in the future. Travel sites, while generally matched by TX2, are particularly difficult for our matching algorithms to handle completely because most sites make use of a custom date picker interface. This interface is rarely recorded correctly by our current framework, and thus interactions with it are not appropriately matched.

Furthermore, because these widgets are often so different across different sites, we would no doubt have difficulty appropriately matching the steps even if the steps were recorded correctly. It seems that it will be necessary to add a mechanism in the future that allows us to provide custom code in the back-end to match these radically different widgets.

Another issue is matching forms consisting of a single textbox and search button, as with the generic search engines and many other search forms on other web sites. With a small corpus and very few values as features, it is difficult to rank the similarities and differences between such forms. We hope that with a larger data set, the differences between forms should become more pronounced and judicious use of thresholds (or perhaps automatically changing the thresholds over time) will solve some of the problem of bad matches. It seems likely, however, that the generic search engines will always match to keyword search forms on other web sites, but this is not necessarily a problem. It could be useful to see the results of the corresponding Google search every time you make use of a web sites' own internal search feature.

Meta-Search Evaluation

We next tested our client-side infrastructure for using the matches retrieved from the web service by manually defining the matches and determining if TX2 could combine them correctly. We first created matches for 10 sites chosen from 6 different categories as listed in Table 2. Overall, TX2 successfully crawled and combined the results from 6 of the 10 web sites on which it was tested. These represented the search engine, library, and shopping sites already popular as meta-search engines. TX2 can create these automatically, allowing users to choose the sites that they want included in personalized meta-search mash-ups without relying on someone else to create an appropriate mash-up for them.

TX2 failed to connect sites for two different reasons, both of which we believe can be overcome in future versions. First, TX2 was unable to detect the repeating result elements on the two video sharing sites in the evaluation. This is not surprising as these results are detected using the technique also used in Sifter as a base, and in an evaluation of that system it was shown to not always be able to find the correct repeating result element [10]. The authors believe that by adding additional structure to the DOM, perhaps by identifying additional repeating patterns in the content of results (instead of only in the DOM structure), this problem could be fixed.

The second problem was that TX2 was unable to automatically submit the forms on both the online shopping sites and social networking sites in our evaluation. These sites used non-standard methods to submit form queries. For example, to submit a search on *ebay.com*, users click a link that calls a Javascript function that does some preprocessing and then loads a new URL. Because the form cannot be submitted by calling the submit() function of the associated form, TX2 cannot currently submit this form. In the future, we plan to leverage CoScripter's recording features to learn how to submit forms based on user interactions. In the *ebay.com* example, TX2 could have learned that the typical interaction pattern is to fill out an input field and then click a certain link, after which a new page loads. The CoScripter "slop" currently being recorded can capture and play back such interactions; they simply need to be integrated as part of TX2's crawling process.

Site 1	Site 2	Result
google.com	live.com	Match
google.com	yahoo.com	Match
live.com	yahoo.com	Match
spl.org	kcls.org	Match
youtube.com	hulu.com	Result Failure
hulu.com	youtube.com	Result Failure
craigslist.org	froogle.com	Match
craigslist.org	amazon.com	Match
facebook.com	myspace.com	Submit Failure
craigslist.org	ebay.com	Submit Failure

Table 2: Results of Meta-Search Evaluation. *Match* means the connection was successful, *Result Failure* means that TX2 was unable to automatically identify the results section after the page submission, and *Submit Failure* means TX2 was unable to automate the submission of the form.

DISCUSSION

TX2 works well at creating the meta-search mash-ups for which it was designed, and suggests a rich area for future research on using interaction histories to help automatically create mash-ups. In this section, we have highlighted some of the limitations of our current implementation, opportunities for future work, and remaining challenges for creating mash-ups automatically from user interactions.

Limitations and Opportunities for Future Work

TX2 demonstrates a new model of mash-up creation in which recorded web interactions are mined to automatically link deep web resources. TX2 currently only creates meta-search mash-ups, but future work may explore how other types of mash-ups could be created automatically. We believe that to truly move the creation of mash-ups from the domain of programmers to everyday users requires more automation. TX2 succeeds for an important class of mash-ups; extending automatic creation to other types of mash-ups is future work.

As with other mash-up systems, TX2 may draw complaints from site owners who object to its repackaging of their content. This is particularly true when TX2 has matched the visual styling to make results from different sites appear to have originated from another site. Meta-search has previously been done at smaller scales in terms of the number of sites, but TX2 has the potential to expand this to every site in the deep web. The authors can imagine several scenarios for addressing these concerns, for example by explicitly noting the source web site next to each result or by allowing sites to opt out of TX2. Unlike traditional crawlers, TX2 is a client-side tool and so exclusion may be harder to enforce. More generally, TX2 provides a service that we believe users will find useful and which existing rules do not adequately cover.

Security and Privacy

To protect privacy, forms on secure sites and forms that include password fields do not need to be recorded. This would help keep the most sensitive information submitted to web

forms private. Even the anonymized version of search queries can cause privacy concerns. AOL released the query logs of 500,000 users in 2006 that revealed information about these users that they likely would have preferred to have remained private (for instance, based on their queries, certain users could be identified as likely to have a certain health condition). TX2 does not need to record the specific text input into web forms, but only needs to recognize when inputs are the same. TX2 could store anonymized versions of the inputs hashed with a secure, one-way hash like MD5. This would add a substantial layer of protection because any adversaries would first need to guess the input that the user provided and then verify that it exists in the repository.

Another option is to enable users to submit only the actions that they performed when executing a query, and to only require this information when the user wishes to find matching forms through the repository. Users would then be making an explicit choice about when to share and when to accept matches, and potentially limit the consequences of this choice by only contributing a small part of their interaction history. As a side effect, the database would need to perform less filtering of the data that is added to it because more of the data would pertain to form interaction.

Regardless of the protections offered, some users may choose not to contribute their interaction histories or query actions to the repository. These users could still use the connections that have been demonstrated by others. Those who don't trust the connections demonstrated by others may still choose to use only the connections that they have personally defined. Contributing connections seems to pose less risk. Exposing that a user has searched for the same information on two resources reveals much less information than revealing what they searched for on those resources.

CONCLUSION

We have shown that mash-ups can be created automatically by mining web interactions, and have presented a new algorithm that uses recorded interactions to connect web forms. We have also presented an implementation of this system that makes these mash-ups easy for users to use and create by augmenting the sites they already visit with results from other related sites. TX2 lets users aggregate data from multiple sources absent any explicit requirements of the web sites.

Acknowledgements

We thank Bin He for his guidance and useful suggestions.

REFERENCES

1. Adar, E., Teevan, J., and Dumais, S. Large scale analysis of web revisitation patterns. In *Proc. of the SIGCHI Conf. on Human factors in Comp. Sys. (CHI '09)*. Boston, Massachusetts, USA, 2009.
2. Bigham, J., Cavender, A. C., Kaminsky, R. S., Prince, C. M., and Robison, T. S. Transcendence: Enabling a personal view of the deep web. In *Proc. of the 13th Intl. Conf. on Intelligent User Interfaces (IUI '08)*. Gran Canaria, Spain, 2008.
3. Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R. C. Automation and customization of rendered web pages. In *Proc. of the 18th ACM Symp. on User Interface Soft. and Tech. (UIST '05)*. Seattle, WA, USA, 2005, 163–172.
4. Chang, K. C.-C. and He, B. Toward large scale integration: Building a metaquerier over databases on the web. In *Proc. of the 2nd Conf. on Innovative Data Sys. Research*. 2005.
5. Doan, A., Domingos, P., and Halevy, A. Y. Reconciling schemas of disparate data sources: a machine-learning approach. In *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of data (SIGMOD '01)*. 2001, 509–520.
6. Dontcheva, M., Drucker, S. M., Wade, G., Salesin, D., and Cohen, M. F. Summarizing personal web browsing sessions. In *Proc. of the 19th ACM Symp. on User Interface Soft. and Tech. (UIST '06)*. New York, NY, USA, 2006, 115–124.
7. Faaborg, A. and Lieberman, H. A goal-oriented web browser. In *Proc. of the SIGCHI Conf. on Human Factors in Comp. Sys. (CHI '06)*. Montreal, Quebec, Canada, 2006, 751–760.
8. Fujima, J., Lunzer, A., Hornbk, K., and Tanaka, Y. Clip, connect, clone: combining application elements to build custom Interfaces for information access. In *Proc. of the 17th ACM Symp. on User Interface Soft. and Tech. (UIST '04)*. ACM Press, New York, NY, USA, 2004, 175–184.
9. Hartmann, B., Wu, L., Collins, K., and Klemmer, S. Programming by a sample: Rapidly prototyping web applications with d.mix. In *Proc. of the 20th Symp. on User Interface Soft. and Tech. (UIST '07)*. Newport, RI, USA, 2007.
10. Huynh, D. F., Miller, R. C., and Karger, D. Enabling web browsers to augment web sites' filtering and sorting functionalities. In *Proc. of the 19th ACM Symp. on User Interface Soft. and Tech. (UIST '06)*. ACM Press, New York, NY, USA, 2006, 125–134.
11. Jung, H., Allen, J., Chambers, N., Galescu, L., Swift, M., and Taysom, W. One-shot procedure learning from instruction and observation. In *Proc. of the Intl. FLAIRS Conf.: Special Track on Natural Language and Knowledge Representation*.
12. Lin, J., Wong, J., Nichols, J., Cypher, A., and Lau, T. A. End-user programming of mashups with vegemite. In *Proc.c of the 13th Intl. Conf. on Intelligent user Interfaces (IUI '09)*. Sanibel Island, Florida, USA, 2009, 97–106.
13. Little, G., Lau, T., Cypher, A., Lin, J., Haber, E. M., and Kandogan, E. Koala: capture, share, automate, personalize business processes on the web. In *Proc. of the SIGCHI Conf. on Human factors in Comp. Sys. (CHI 2007)*. 2007, 943–946.
14. Madhavan, J., Halevy, A., Cohen, S., Dong, X., Jeffrey, S. R., Ko, D., and Yu, C. Structured data meets the web: A few observations. *IEEE Computer Society: Bulletin of the Technical Committee on Data Engineering*, *31*, 4 (2006), 10–18.
15. Miller, R. C. and Myers, B. Creating dynamic world wide web pages by demonstration (1997).
16. Mukherjee, S., Yang, G., Tan, W., and Ramakrishnan, I. Automatic discovery of semantic structures in html documents. In *Proc. of the Intl. Conf. on Document Analysis and Recognition (ICDAR '03)*. 2003.
17. Piggy bank. http://simile.mit.edu/piggy-bank/. Accessed April 2009.
18. Pilgrim, M., ed. *Greasemonkey Hacks: Tips & Tools for Remixing the Web with Firefox*. O'Reilly Media, 2005.
19. Raghavan, S. and Garcia-Molina, H. Crawling the hidden web. In *Proc. of the Twenty-seventh Intl. Conf. on Very Large Databases (VLDB '01)*. 2001.
20. Selberg, E. and Etzioni, O. Multi-service search and comparison using the metacrawler. In *Proc. of the 4th Intl. World Wide Web Conf.*. Darmstadt, Germany, 1995.
21. Solvent. http://simile.mit.edu/solvent. Accessed April 2009.
22. Toomim, M., Drucker, S. M., Dontcheva, M., Rahimi, A., Thomson, B., and Landay, J. A. Attaching UI enhancements to websites with end users. In *Proc. of the ACM Conf. on Human Factors in Comp. Sys. (CHI 2009)*. Boston, MA, USA, 2009.
23. Wong, J. and Hong, J. I. Making mashups with marmite: towards end-user programming for the web. In *Proc. of the SIGCHI Conf. on Human factors in Comp. Sys. (CHI '07)*. San Jose, CA, USA, 2007, 1435–1444.
24. Yahoo! pipes. Yahoo! Inc.. http://pipes.yahoo.com/. Accessed February 2009.