From Web Summaries to Search Templates: Automation for Personal Tasks on the Web

Lubomira A. Dontcheva

March 8, 2009

As the amount of content delivered over the World Wide Web grows, so does the consumption of information. And although advancements in search technologies have made it much easier to find information on the Web, users often browse the Web with a particular task in mind, such as arranging travel plans, making purchases, or learning about a new topic. When users have a task in mind, they are often concerned not only with finding but also with collecting, organizing, and sharing information. These types of browsing sessions, which we call *exploratory Web research* sessions, typically last a long time, span several sessions, involve gathering large amounts of heterogeneous content, and can be difficult to organize ahead of time, as the categories emerge through the tasks themselves [20]. Current practices for collecting and organizing Web content such as using bookmarks or tabs, collecting content in documents, storing pages locally, or printing them out [12] require a great deal of overhead as pages must be saved manually and organized into folders, which distracts from the real task of analyzing the content and making decisions.

In our work we break out of the webpage paradigm and consider the individual pieces of content inside of the webpage to be the basic unit that must be collected, as it is the information inside the webpage that is of most importance. If the goal is to let people more easily accomplish their information tasks, then tools must support the manipulation of information, not webpages. In this chapter we discuss the Web Summaries project, which was first published in two papers [6, 5].

There are a few examples of systems that give users access over the content inside of a webpage, such as HunterGatherer [19], Internet Scrapbook [22], and C3W [8], but it is the Semantic Web that promises to truly transform the way people manipulate information. Unfortunately, the Semantic Web remains unrealized largely because it requires a large collaborative effort in defining an appropriate data representation and adopting that representation. Content providers are not yet willing to invest in embedding semantic information into the existing Web and coordinating their efforts with others.

We take advantage of three trends in the World Wide Web – the growing number of structured webpages, the vast rise in online collaboration, and pervasive search technologies – and present a new approach for collecting and

organizing Web content in a set of semi-automatic interaction techniques and algorithms that allow people to not only collect and organize Web content more quickly and easily but also enable them to build a form of the Semantic Web as they accomplish their own tasks.

1 The User Experience

We designed the interface of the Web Summaries system with the goal of making the process of collecting information as unobtrusive as possible and allowing the user to focus on the task at hand rather than worry about organizing and keeping track of content. The system is implemented as an extension to the Firefox browser and is presented to the user through a toolbar (see Figure **??**). The toolbar includes four buttons and a checkbox. The "Start" button opens the "summary" window that displays a visual summary of the content gathered thus far. The "Select" button initiates the selection mode and enables the user to select and label pieces of Web content. The "Add Page" button allows the user to automatically add the content found by an extraction pattern to the summary. This button is enabled only when there is a matching pattern that can be used to extract content automatically. The "Add Linked Page(s)" button initiates a different selection mode in which the user can select any number of hyperlinks to add the content from the linked pages to the summary directly and simultaneously. A checkbox specifies whether to visually outline the elements found by an extraction pattern on a new webpage. If it is checked, the elements found are outlined in purple. The summary window (shown in Figure 1a) has buttons for opening and saving summaries and a menu for changing the layout template used to compose the summary.

1.1 Sample user scenario

To help explain the interface, we describe an example user scenario and show the steps taken by the user to create a summary. In particular, we describe the steps a user takes in planning a night out in the city of Seattle.

The user starts the browsing session by visiting `www.nwsource.com` and looking through restaurant listings. When he finds a restaurant he wants to save to his summary, he presses the "Select" buton, which enables the selection mode. This mode disables the webpage's default functionality to allow the user to select page elements, such as a paragraph or an image. As the user moves the cursor over the webpage, the page element directly underneath the cursor is outlined in red. To select an item, the user clicks with the left mouse button. The outline becomes purple, and the item remains highlighted while he selects other elements. The user also assigns a label to each selected element with a right-button context menu. This label assignment step is necessary because the summary layout templates are defined with respect to these labels. To finish the selection mode and save the selected elements to the summary, the user turns off the "Select" button. The system stores the selected elements locally and

builds an extraction pattern for the selected elements in that page. The user can view the collected content as a summary at any time by going back to the summary window and selecting a layout template, such as a calendar, map, or grid.

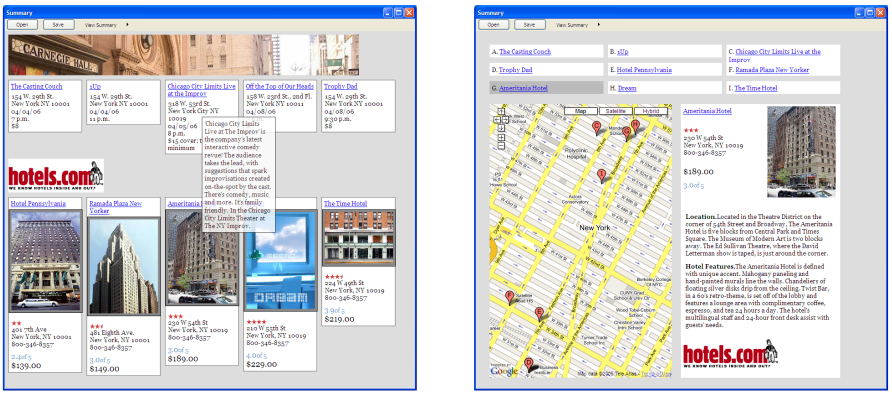
When the user finds a new restaurant he wants to save, all he has to do is press the "Add Page" button and all of the same content is automatically added to the summary. He can also add several restaurants to the summary simultaneously by selecting hyperlinks. To add content simultaneously, the user presses the "Add Linked Page(s)" button, which enables the hyperlink selection mode. To finish selecting hyperlinks, the user turns off the "Add Linked Page(s)" button, and the content from all the linked pages is simultaneously added to the summary.

The user can select new page elements for pages he has already visited, without returning to the page where he first selected the elements. For example, to add the address to all the already gathered restaurants, he can select the address on any restaurant page. He again initiates the selection mode with the "Select" button and selects the address. The elements corresponding to previously selected page elements are highlighted in purple. When he is finished, the summary is automatically updated to include the addresses of all the events he has already gathered. Because the saved restaurants now include an address, the user can see where they are located using the map layout template (Figure 1b).

In addition to gathering information about restaurants, the user can collect any other type of information, such as movies or current events in the area. When he goes to a new website and wants to add content, he must first specify which content for that website is relevant. This is necessary because this system gathers content using the structure of a given webpage, as we describe in the next section, and different websites have different structures. If, however, he has been to this website in a previous browsing session and has already specified important elements, he can use the same specification and automatically collect new content. Since the labels assigned to data from different websites are a constrained set, all the content can be presented uniformly in one summary. The user can add content from any number of sites and go back and forth updating the summary. All summary elements are linked to the original webpages, and the user can navigate directly from the summary and return to those webpages.

1.1.1 Relations

As he looks through his set of saved restaurants, the user decides to check the review website `yelp.com` and look for reviews. He finds reviews about the restaurant Brasa, which is in his collection and decides to add them to his collection. He again selects the parts of the webpage of interest and adds them to his collection. Since the content extracted from `nwsource.com` and `yelp.com` refers to the same restaurant, the user can relate them together. To create a relation, he draws a line from the name, "Brasa," collected from `nwsource.com` to the name, "Brasa Restaurant," collected from `yelp.com` (Figure **??**a). The system responds by visually joining the extracted content and displaying "Con-



(a) (b)

Figure 1: **(a)** The grid layout template places all the content in the database on a grid in groups, according to extraction pattern. **(b)** With the map layout template, the user can view the content with respect to a geographical map. The template filters the database and presents only content that includes an address.

necting nwsource.com to yelp.com." Now, when the user adds a new restaurant from `nwsource.com` to his collection, the corresponding review from `yelp.com` is automatically retrieved and added to the collection. He can also collect hyperlinks pointing to potentially interesting restaurants at `nwsource.com`, and the reviews for those restaurants will be automatically collected from `yelp.com`).

Upon inspecting his collection of restaurants, the user decides that he might need to take the bus to his dinner destination. He visits the website `metrokc.gov` and finds bus information for the downtown area. He adds the bus information to his collection and interactively connects the neighborhood of Brasa — "downtown" — to the bus route information). Now when he collects a new restaurant, the system will automatically collect reviews for the restaurant from `yelp.com` and bus schedules from `metrokc.gov`. To retrieve the bus schedules for all the restaurants he has already collected, the user clicks on the "Update All" button, and the corresponding bus schedules are retrieved automatically.

If the retrieved results are not the correct ones, the user can click on the icon in the bottom right corner and see any other content that was retrieved. For example, if he clicks on the icon for the "Brasa Restaurant" from `yelp.com`, he will see a lit of other "Brasa" restaurants that were retieved from `yelp.com`, such as "Pollo Brasa y Sazon," "Pollo a La Brasa Vermont," "Fogo E Brasa," or "Pollos A La Brasa Marion." The user can at any time pick the best content from this list or return to a website to collect new information.

1.1.2 Cards

As the user collects more content, his collection space quickly fills up. The user can address the growing clutter by creating a specialized card that displays

only the information of interest. To create a card, the user clicks on the "New Card" button and opens a canvas for card design. He first draws the outline of the card and then draws containers inside of the card. He places content from his collection directly into the containers. The user can resize and reposition the containers until he is satisfied. He clicks "Done," names the card, and can now view all the related collected content as personalized cards. Cards can be designed for any size or purpose and can contain information from any number of source webpages. They also include hyperlinks to the original webpages so that the user can at any time return to the original content.

1.1.3 Search templates

In addition to collecting content by visiting actual websites, the user can also collect content through keyword queries and a search template. Thus, a user can go directly from a query term, such as "seafood," to a series of cards filled with content from multiple sources. The system collects seafood restaurants and any related content. Search results are considered only temporary and are not part of the user's collection. Thus, they are displayed separately, below the existing collection. The user can promote any search result to the actual collection by clicking on the "+" button in the upper left corner. He can also delete a card with the "x" button. This way the user can quickly scan through many restaurants and identify good options.

A search template is defined implicitly by a user-defined card. To create a search template, the user clicks on the "New Search Template" button, selects a card, and can optionally add additional websites or relations to be part of the template. A search template allows the user to package all of the work he has already done in collecting and associating content and use it to find new content more efficiently. Figure 2 shows a query for "chocolate cake" with the "recipes" search template, which retrieves and reformats recipes from `allrecipes.com` and `cooking.com`. Figure 3 shows several queries for different types of cars. Each card includes car specifications and reviews from `autos.msn.com` and `edmunds.com`.

The Web Summaries experience separates the content presented on the Web from its structure, presentation, and source, and allows the user to create personal summaries of content. This type of interface can lower the overhead of gathering and managing data and allow the user to focus on the task at hand.

2 System Design

We first describe the overall system and then explain the technical details for each part. The system includes five components: an extraction pattern repository, a data repository, a set of pre-defined layout templates, user-defined cards, and a set of search templates. The *extraction pattern repository* contains *patterns* defined from the user-selected page elements. When the user visits a new page, the patterns in the extraction repository are compared with the page. If

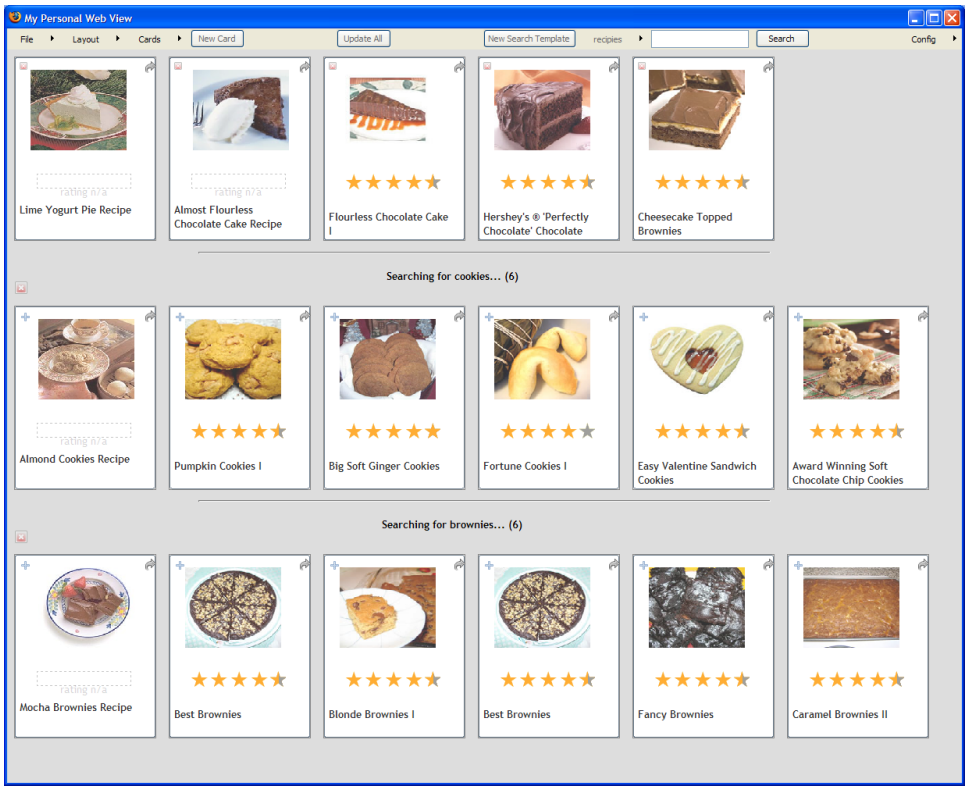


Figure 2: With the recipe search template, the user collects recipes from `cooking.com` and `allrecipes.com`. Here the user has a collection of five cards and has made two queries, one for "cookies" and and another for "brownies". The system automatically collects, extracts, and displays relevant recipes.

there are matches, the user can add the matching content to the data repository. The *data repository* includes relation trees and the summary database that holds all of the content collected by the user according to the source webpage and semantic tags of the webpage elements. We refer to each piece of content collected by the user as a *webpage element*. Each webpage element is associated with a semantic tag, such as name, address, date, time, etc. All webpage elements collected on the same webpage form a record in the data repository. The data repository also holds *relations*, which specify relationships between tags in different records. Records that are related in this way form a *relation tree*.

The *layout templates* filter the database to create summary views. The user can adapt the templates by authoring cards. A *card* defines which webpage elements within a relation tree should be displayed and their visual arrangement.

The user can collect data by visiting webpages or through search templates. A *search template* includes a set of websites and possibly relations for those websites. When the user types a keyword query, the search template queries each of the websites with the keyword, extracts content from the search results

with extraction patterns, triggers the collection of related content, and displays them as a series of cards.

2.1 Gathering content semi-automatically

The page-element selection interface uses the Document Object Model (DOM) structure to provide a mechanism for selecting content. When the user initiates the selection mode, the typical behavior of the webpage is frozen, and the browser mouse event handlers are extended to allow the user to select pieces of the DOM hierarchy. As the user moves the cursor and clicks, the DOM nodes directly underneath the cursor are highlighted. Once the user has selected a set of nodes, the system generates an extraction rule for each selected node. The *extraction rule* consists of the selected node, the path from the root of the document to the selected node, and the user-assigned label. The path enables finding analogous elements in documents with similar structure. Structural extraction rules are also known as XPATH queries. The extraction rules rely on consistent structure. Thus, if the structure changes, the user will have to go back and re-specify extraction rules. Gibson et al. show that template material changes every 2 to 3 months; however, they give few details on the types of changes. To evaluate the performance of structural extraction patterns over time, we conducted a five-month study of webpage changes. Please see [] for details.

In addition to the *structural* extraction rules just described, the system also provides content-based rules. *Content-based extraction rules* collect content from new webpages by matching text patterns instead of structural patterns. To specify a content-based rule, the user selects an element and labels it not with a keyword, as he does with the structural rules, but with text from the selected element that should be matched in analogous elements. For example, to only collect articles by author "John" the user selects an article and its author, chooses "semantic rule" from the right-button context menu, and types "John." A content-based rule first tries to find matching content using the structural path, but if it is not successful, the rule searches the entire document. It finds matching content only if the node types match. For example, if the rule finds the word "John" in a `<table>` node and the selected node defining the rule is in a `<p>` node, the rule will fail. This limits the number of spurious matches and ensures consistency in the gathered content. The effectiveness and power of content-based rules, when possible, was shown by Bolin et al. [3] in Chickenfoot.

The user may specify any number of extraction rules for a given page. As those rules should always be applied together, the system collects the extraction rules into *extraction patterns* and then stores them in the extraction pattern repository. An extraction pattern can include any number of extraction rules and can be edited by the user to add or remove rules. For example, the user might care about the name, hours, and address of a restaurant. The system creates an extraction rule for each of these elements and then groups them together so that for any new restaurant page, it searches for all three page elements in concert.

When the user visits a webpage, each available extraction pattern for that Web domain is compared with the DOM hierarchy, and the pattern with the highest number of matching rules is selected as the matching pattern. If the user chooses to store the matched content, the webpage is stored locally, and the elements found by the matching pattern are added to the summary database. When the user selects hyperlinks to collect content from multiple pages simultaneously, the system loads the linked pages in a browser not visible to the user, compares the pages with the extraction patterns, and adds all matching elements to the database. If an extraction pattern does not match fully, it may be because some of the content is absent from the page, or because the structure of the page is slightly different. In these cases, the user can augment the extraction pattern by selecting additional elements.

Although the growing use of webpage layout templates for formatting and organizing content on the Web makes it possible to automate collecting information from the Web, this automation comes at a cost. The automatic extraction is sensitive to the structure of HTML documents and depends on a sensible organization to enable the selection of elements of interest. If the structure does not include nodes for individual elements, the user is forced to select and include more content than necessary. On the other hand, if the structure is too fine, the user must select multiple elements, adding overhead to the selection process. Most websites that do use templates tend to use templates with good structure, because good structure makes it easier to automate webpage authoring.

2.2 Retrieval using relationships

When the user connects content collected from different webpages, he creates a relation. We define a *relation* as a directed connection from tag_i from $website_A$ to tag_j from $website_B$. For example, when the user draws a line between the names of the restaurants, he creates a relation from the "name" tag on Northwest Source to the "name" tag on Yelp. When the user connects restaurants and buses, he creates a relation from the "area" tag to the "route" tag. All relations are stored in the data repository and are available to the user at any time. Webpage elements that are associated through relations form a relation tree.

When the user collects content from a new webpage, the system checks for relations that connect any of the collected webpage elements to other websites. When such relations exist, the system uses them to generate new search queries and limits the search results to the website specified in the relation. For example, when the user collects information about the restaurant "Nell's," the system generates two queries. To collect restaurant reviews it generates a query using the "name" tag, i.e. "Nell's," and limits the search results to `yelp.com`. To collect bus schedules the system generates a query using the "area" tag, i.e. "Green Lake," and limits the search results to the bus website, `metrokc.gov`.

To define this process more formally, the execution of a relation can be expressed as a database query. For a given relation r, where $r = website_A.tag_i \rightarrow website_B.tag_j$, one can express the process of automatically collecting content

for any new data record from $website_A$ for tag_i as a `JOIN` operation or the following SQL pseudo-query:

`SELECT * FROM` $website_B$ `WHERE` $website_B.tag_j = website_A.tag_i$

Since the Web is not made up of a set of uniform databases, we use a number of different techniques to make this query feasible. We use the Google Search AJAX API to find webpages within $website_B$ that are relevant. To extract content from each of the search results, we employ the user-defined extraction patterns. Finally, we designed a number of heuristics to compute a similarity metric and rank the extracted search results. The system displays only the highest ranked extracted search result to the user but makes the remaining search results available.

In the current implementation, the system extracts content from only eight search results because the Google AJAX Search API limits the search results to a maximum of eight. For all of the examples in this dissertation, eight search results are sufficient and limit the delay in collecting information. For very common keywords, however, collecting eight search results is not sufficient. For example, searching for "Chili's" will yield many instances of the restaurant chain. For those types of queries narrowing the search through one of the approaches mentioned above would be necessary.

Our approach for collecting related content is limited to websites that are indexed by general search engines. There are many websites, such as many travel websites, that are not indexed by search engines because they create webpages dynamically in response to user input. To handle these dynamic webpages, in subsequent work we hope to leverage research into macro recording systems such as WebVCR [1], Turquoise [16], Web Macros [18], TrIAs [2], PLOW [13], and Creo [7]. These systems allow users to record a series of interactions, store them as scripts, and replay them at any time to retrieve dynamic pages. Recent research on retroactive macro recording could be even more applicable for Web Summaries [11]. Madhavan et al. [15] are developing information retrieval approaches to this problem that do not require user intervention.

The search process introduces ambiguity at two levels, the query level and the search result level. The system must be able to formulate a good query so that it can retrieve the relevant content. It must also be able to find the correct result among potentially many that may all appear similar. Both of these forms of ambiguity pose considerable challenges and are active areas of research. Liu et al. [14] pose the query formulation problem as a graph partitioning problem. Dong et al. [4] propose propagating information across relations to better inform similarity computation. Next, we describe how we address these two types of ambiguity.

2.2.1 Query formulation

We formulate two keyword queries in parallel. One query includes only the extracted text content, and the other includes the extracted content and the tag associated with the content. These two queries are usually sufficient to collect the appropriate result within the top eight search results. Sometimes, however,

queries may include too many keywords, and the search results are irrelevant or cannot be extracted. In such cases, We employ heuristics to reformulate the query. If characters such as '/','-','+', or ':' appear in the text, we split the string whenever they appear and issue several queries using the partial strings. We found this approach particularly effective for situations in which something is described in multiple ways or is part of multiple categories. For example, a yoga pose has a Sanskrit name and an English name. Querying for either name returns results, but querying for both does not, as the query becomes too specific. Other approaches for reformulating queries include using the semantic tag associated with the webpage element or using additional webpage elements, such as the address, to make the query more or less specific. With an interactive system, processing a large number of queries can be prohibitive due to the delay caused by the search and extraction process. We focus on finding good heuristics that quickly retrieve results that are close to the desired content. If the system fails to find a good search result, the user can always go to the website and collect the content interactively.

2.2.2 Search result comparison

For each query we extract the first eight search results and rank the extracted content according to similarity to the webpage content that triggered the query. To compute similarity we compare the text of the extracted webpage elements using the correspondence specified in the relation that triggered the search. For example when collecting content for the "Ambrosia" restaurant from `nwsource.com`, the system issues the query "Ambrosia" limiting the results to the `yelp.com` domain. The search results include reviews for the following establishments: "Ambrosia Bakery" (in San Francisco), "Cafe Ambrosia" (in Long Beach), "Cafe Ambrosia" (in Evanston), "Ambrosia Cafe" (in Chicago), "Ambrosia on Huntington" (in Boston), "Ambrosia Cafe" (in Seattle), and "Caffe Ambrosia" (in San Francisco). Because the relation between `nwsource.com` and `yelp.com` links the names of the restaurants, we compare the name "Ambrosia" to all the names of the extracted restaurants. We compare the strings by calculating the longest common substring. We give more weight to any strings that match exactly. For all seven restaurants in this example, the longest common substring is of length eight; thus, they receive equal weight. Next, we compare any additional extracted elements. We again compute the longest common substring for corresponding webpage elements. In this example, we compare the addresses of the extracted restaurants and compute the longest common substring for each pair of addresses, resulting in a ranking that places the Seattle restaurant "Ambrosia Cafe" as the best match to the original content. We display the highest ranked extracted content but provide all of the extracted content to the user so that he can correct any errors. The highest ranked content is marked as confident when multiple webpage elements match between websites.

The problem of retrieving related information from multiple different sources is described in the database community as data integration [10]. Data integration is the problem of combining data residing in different sources and providing

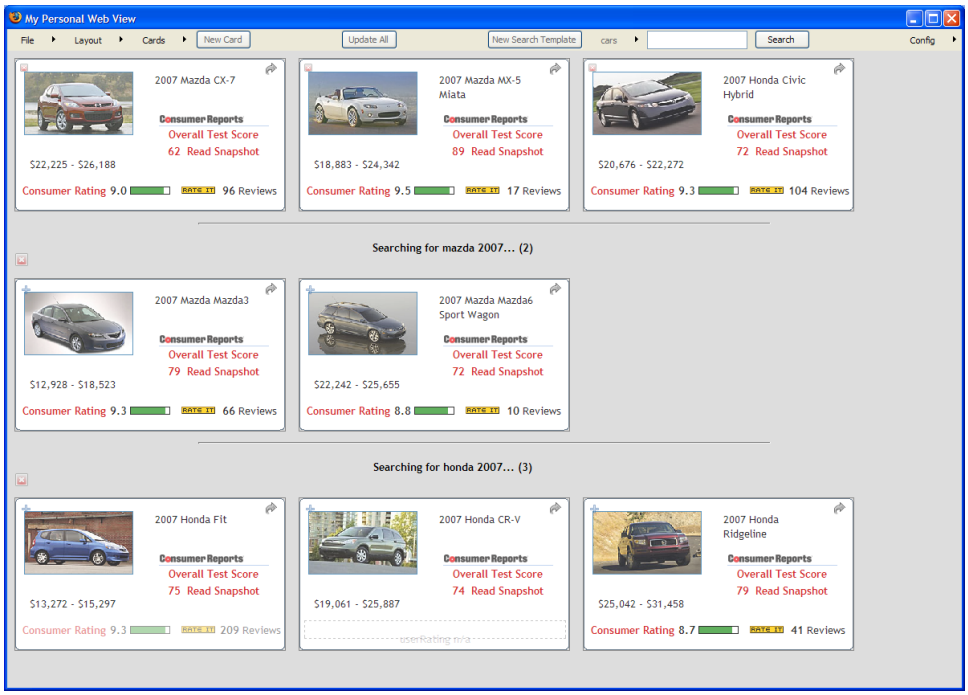


Figure 3: The user is shopping for cars and using a car search template to find new cars. He has three cards in his collection and has made two queries: "mazda 2007" and "honda 2007." Each card includes car reviews from `autos.msn.com` and `edmunds.com`.

the user with a unified view of these data. The difficulty in data integration lies in forming mappings between heterogeneous data sources that may include different types of data and defining one single query interface for all of the sources. This problem emerges in a variety of situations both commercial (when two similar companies need to merge their databases) and scientific (combining research results from different bioinformatics repositories). With the growth of the Web, database researchers have shifted their focus towards data integration of unstructured Web content and its applications to Web search [15]. My work is complementary to database research in that it offers interactive techniques for data integration on a personal scale. We provide an interface that allows users to specify mappings between different data sources, i.e. websites, and then use these mappings to automatically extract content from the Web.

Finally, the current implementation allows the user to specify only one-to-one relations. In some situations a one-to-many relation is more appropriate; for example, if the user is interested in collecting academic papers and wants to collect all of the papers written by each of the authors for any given publication. The system can actually query for all of the papers by a given author but it is not designed to let the user view all elements of the collection as relevant. In future work, we plan to explore other types of relations and also introduce

transformations into the relations.

2.3 Summary composition

The database organizes the webpage elements according to the user-assigned label, the page where it was found, and the extraction pattern used to collect it. Since the same set of labels applies to all webpages, layout templates that filter the database to create summaries use the labels rather than the specific HTML content.

A layout template consists of placement and formatting constraints. The *placement constraints* specify how the data should be organized in the summary. For example, a placement constraint can specify that all content with the label "name" be placed in a list at the top of the document. The position of each element can be specified in absolute pixel coordinates or be relative to previously placed elements. For relative placement, the browser layout manager computes the final content position; for absolute placement, the template specifies all final element positions. Placement constraints can be hierarchical. For example, the template designer can specify that content collected from the same webpage be grouped into one visual element and that such groupings be organized in a list. Although the hierarchy can have any depth, in practice we have found that most layout templates include placement constraint hierarchies no deeper than two or three levels. *Formatting constraints* specify the visual appearance of the elements, such as size, spacing, or borders.

Each layout template can also specify mouse and keyboard interactions for the summary. For example, when the user moves the cursor over an item in the calendar, a short description of the event is presented. When the user clicks on the item, a detailed view of that item is displayed in a panel next to the calendar.

The layout templates are implemented with Javascript and Cascading Style Sheet (CSS) style rules. To create the summary, the system loads an empty HTML document and dynamically populates the document with the DOM nodes in the database using the placement constraints of the current layout template. Each node is wrapped in a `<div>` container, and the container's class attribute is set to the label stored with the node. This allows us to specify some of the formatting constraints with CSS style sheets.

We provide "save" and "load" functionality, which makes it possible to share a summary and any specified extraction patterns. The user can share the database without sharing all the locally stored HTML documents, making summaries like those shown in this paper no bigger than 500KB. Since the system is implemented as a browser extension, a collaborator can install the extension, load an existing summary and its corresponding extraction patterns, and continue the research session.

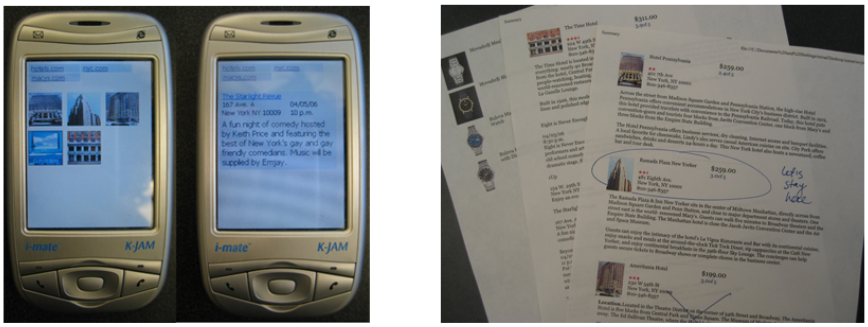


Figure 4: **(a)** The PDA layout template is designed for a small-screen device so that the user can take the summary anywhere. The layout separates the content into tabs according to website and provides detailed views when the user clicks on an item. **(b)** The print layout template formats the content so that it can be printed.

2.4 Layout templates

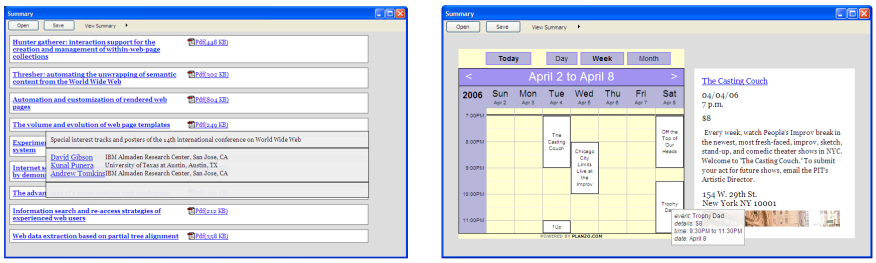
To demonstrate different possibilities for summarizing Web content we implemented a set of layout templates. We present two types of layouts: table-based and anchor-based. The table-based layouts organize the content in a grid, and the anchor-based layouts relate the content through a graphical element.

The *grid layout template* (Figure 1a) places webpage elements found on the same page into one visual element, a box, and arranges these elements according to an extraction pattern. If there is only one pattern specified for a website, as in Figure 1a, the content appears to be arranged according to website. Webpage elements labeled as "description" are available to the user through mouse rollovers.

The *PDA layout template* (Figure 4a) separates the content into tabs to make it more accessible on a small-screen device. Each tab contains a list of the elements collected from the same website. The webpage elements labeled as "name" or "image" are displayed in a list on each tab. When the user clicks on a name or image, the elements found on the same webpage as the clicked item appear. For example, on the PDA on the right in Figure 4a the user clicked on the "Starlight Review" and is now viewing details about this event.

The *print layout template* (Figure 4b) organizes content such that it can be placed on paper. It resizes images and condenses the content to minimize the number of printed pages.

The *text layout template* (Figure 5a) is designed for text-intensive tasks, such as literature surveys. The template organizes the names — or, in the case of Figure 5a, the paper titles — in a list. The user can click on each title to see more information, such as the paper authors or abstract. If present, a link to the document is placed with the title.



(a) (b)

Figure 5: **(a)** The text layout template is designed for collecting text content, as is common in a literature search. The paper titles are placed in a list, and further details for each paper, such as the authors and abstract, are available through mouse interactions. **(b)** With the calendar layout template, the user can view content with respect to time. When the user clicks on an event, details about the event are displayed in a panel next to the calendar.

We present two anchor-based layouts, a map and a calendar. An anchor-based layout allows layout of items with respect to a common element. To create this relationship we analyze the collected content. Any element with the label "address" is parsed to find the actual address, while any element with the label "time" is parsed to find the date and time. As in previous work [21], we use heuristics to find the address, date, and time within the selected nodes. While these heuristics are not capable of finding an address in a whole document, they are sufficient for finding the address in the nodes typically selected.

The *map layout template* (Figure 1b) has three parts, a list of names at the top, a map in the bottom left, and a detail-view panel in the bottom right. The user can click on a name on the list to see its details in the detail-view panel and its location on the map. The user can also interact with the map and navigate the content geographically. Unlike the other templates, the map layout displays only the content that includes an address. To create a map we use the Google Maps API [9].

The *calendar layout template* (Figure 5b) displays a calendar on the left and a detail-view panel on the right. Content that includes a date and time is placed in the calendar, and the user can navigate between the day, week, and month view and click on events in the calendar to see more details about each event. Similar to the map layout template, the calendar layout template filters the database and presents only content that includes a date and time. The implementation of the calendar is provided by the Planzo API [17].

2.5 Authoring cards

The user can view his collection of Web content through cards. A card imposes a uniform design on content that may come from many different websites and may initially appear very different. It defines which content should be displayed and how the content should be organized visually. Recall that the user's content

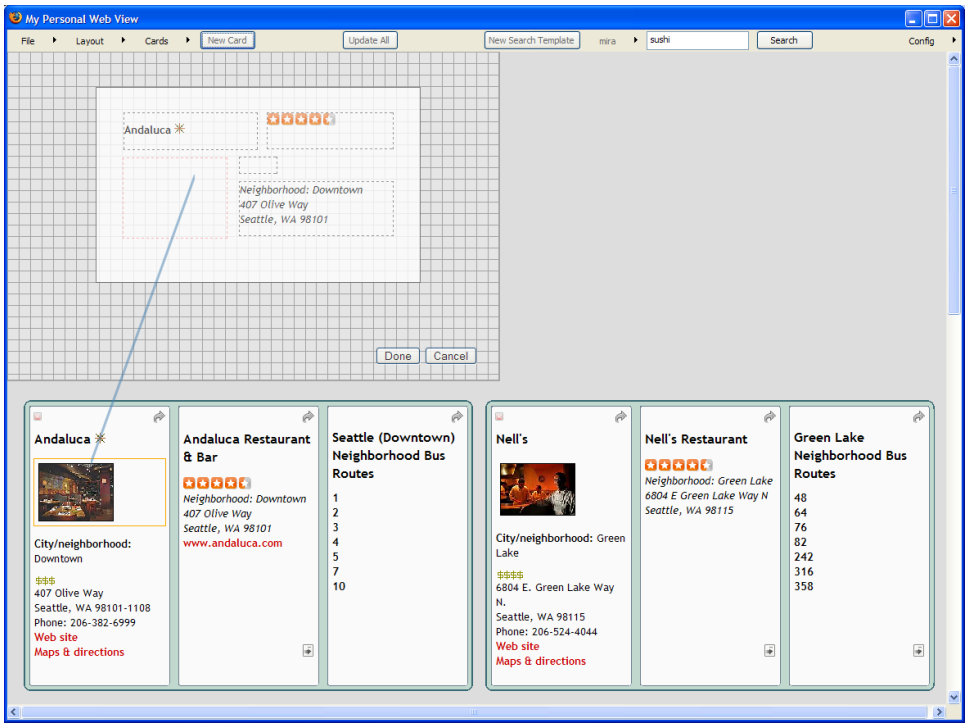


Figure 6: To design a new card, the user opens a canvas and begins drawing. He first draws the outline of the card and then draws containers. To place content in the containers, the user draws a line from the webpage element to the container. Here, the user adds an image to the card.

collection is stored in the data repository and is accessible through relation trees. It is the relation trees that specify which records in the data repository are related. In database terminology, a card can also be described as defining a view on the relation trees in the data repository — i.e., it lists the tags for the data that should be displayed in the card. For example, a card can include the name and address of a restaurant. Or it can also include pricing, rating, and images. The system includes a default card, which displays all records and webpage elements in the relation tree irrespective of the tags associated with the webpage elements. The user can use an interactive editing tool to create new cards at any time. Cards are persistent, can be reused, and shared with others.

To create a new card the user clicks on the "New Card" button, which opens a canvas directly in his collection of Web content (see Figure 6). The card designer is tightly integrated with the collection space so that the user can quickly and easily merge content from different websites without switching windows or views. The user first draws the outline of the card and then proceeds to create containers that hold the webpage elements. To assign data to a container, the user clicks on a webpage element and drags a line to the container. The data is

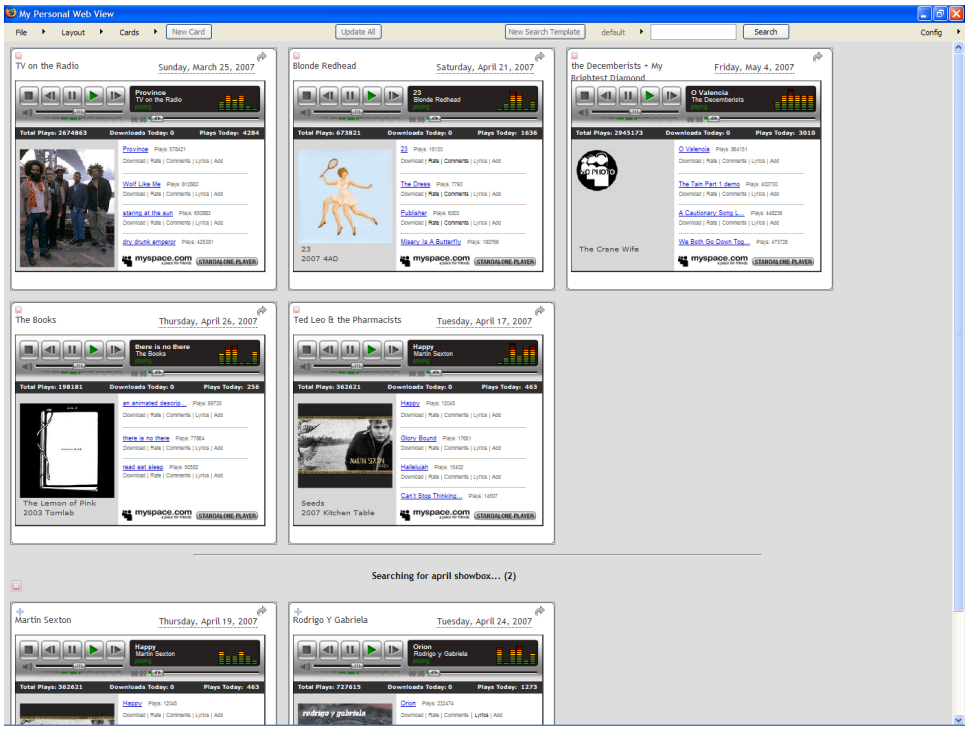


Figure 7: The user relates `upcoming.org` to `myspace.com` to automatically collect music samples for upcoming shows.

automatically resized to fit the size of the container. Each container is associated with the tag of the webpage element it contains and the element website. If the element was not marked as confident during the automatic ranking process, it is rendered semi-transparent to alert users that they may want to confirm the information by clicking on it to go to the source webpage. When the user is finished creating containers and assigning data, he clicks on the "Done" button, and the system transforms his drawing into a template for organizing and displaying Web content, a card. The user can at any time edit the card and add or remove content. Currently, the cards are presented in a grid, but they could also be manually organized into piles or arranged visually on a map or calendar. The authoring principles behind the card designer can also extend to authoring layout templates that organize the cards.

Figure 7 shows a collection of upcoming shows. In this example the user has related concerts from `upcoming.org` with band webpages at `myspace.com`. Whenever the user adds a new concert to his collection, the system automatically collects music samples. The music samples are embedded in a music player, and because the player is just another HTML object, a Flash object, one can extract it just like any other webpage element. The music player retains full functionality, and the user can press the "play" button on the control to listen

to the music.

Cards can be interactive in nature, encoding interactions specific to the type of data that is collected. The card authoring tool does not currently provide capabilities for specifying interactions. We could expose a scripting interface and allow the user to specify any type of card interactions, but since Web Summaries is designed for the novice we chose to remove all scripting from the interactions.

2.6 Template-based search

Search templates combine the user designed cards and relations as a basis for a new type of search interface that targets the keyword query to appropriate domains and organizes the search results in a visual summary. The user can thus bypass visiting webpages directly and collect content through a search template. For example, if the user wants to find vegetarian restaurants but does not know where to start, he can simply query with the word "vegetarian" directed towards the data underlying the restaurant card. More formally, a *search template* includes a set of websites and any associated relations. When a user types a query to the search template, the system sends the query to a general search engine, in this case through the Google Search AJAX API, limiting the search results to the list of websites defined in the template. For each search result, the system extracts content using predefined extraction patterns and triggers any relations that are in the template to collect additional content. Due to limitations on the number of search results provided by the Google Search AJAX API, for each query/website pair, the system processes only eight search results. The user can also modify the search template by adding additional websites to be queried and relations to be triggered. Extracted search results are presented to the user as a set of cards. These are initially considered temporary, indicated by being displayed below the main collection of cards. The user can promote a search result to the actual collection or he can delete all of the search results for a given query.

The success of template-based search lies in the ability to extract semantic information from webpages. Although semantic extraction is only in its infancy, we believe that it will only grow in the coming years, and template-based search is an example of the powerful new applications that will take advantage of machine-readable webpages.

3 Discussion

The interaction techniques we present enable users to transform the Web into their own personal Web that provides personalized access to the information they care about in the form of their choosing. First, we take advantage of the growth in template material on the Web and design semi-automatic interactive extraction of content from similar webpages using structural and content **extraction patterns**. Second, we employ **layout templates** and user labeling to create rich displays of heterogenous Web content collections. Third, we

use search technology for proactive retrieval of content from different related websites through user-defined **relations**. Fourth, we let users define their own personalized and aesthetic views of heterogenous content from any number of websites through **cards**. And finally, we introduce a new **template-based search** paradigm that combines the user-defined relations and cards into a search template. Search templates present a goal-driven search mechanism that creates visual personalized summaries of the content users need to accomplish their task. As users accomplish their goals with these tools, they also produce artifacts, such as the extraction patterns and relations, that are persistent and sharable. These artifacts can be stored in a public repository and reused by others. Furthermore, they can be added to the websites where they originated, thereby enhancing the existing Web with semantic information, such as relationships between different websites or the types of content available in a particular webpage. If popularized by an online community, the ideas we present here can help realize a machine-readable World Wide Web.

References

[1] Vinod Anupam, Juliana Freire, Bharat Kumar, and Daniel Lieuwen. Automating web navigation with the webvcr. In *Proc. of the WWW conference on Computer networks*, pages 503–517, 2000.

[2] Mathias Bauer, Dietmar Dengler, and Gabriele Paul. Instructible information agents for web mining. In *Proc. of the IUI*, pages 21–28, 2000.

[3] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In *Proc. of UIST*, pages 163–172, 2005.

[4] Xin Dong, Alon Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *Proc. of the SIGMOD*, pages 85–96, 2005.

[5] Mira Dontcheva, Steven M. Drucker, David Salesin, and Michael F. Cohen. Relations, cards, and search templates: user-guided web data integration and layout. In *Proc. of UIST*, pages 61–70, 2007.

[6] Mira Dontcheva, Steven M. Drucker, Geraldine Wade, David Salesin, and Michael F. Cohen. Summarizing personal web browsing sessions. In *Proc. of UIST*, pages 115–124, 2006.

[7] Alexander Faaborg and Henry Lieberman. A goal-oriented web browser. In *Proc. of the SIGCHI*, pages 751–760, 2006.

[8] Jun Fujima, Aran Lunzer, Kasper Hornbæk, and Yuzuru Tanaka. Clip, connect, clone: combining application elements to build custom interfaces for information access. In *Proc. of UIST*, pages 175–184, 2004.

[9] Google Inc. http://www.google.com/apis/maps/.

[10] Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Data integration: The teenage years. In *VLDB*, pages 9–16, 2006.

[11] Darris Hupp and Robert C. Miller. Smart bookmarks: automatic retroactive macro recording on the web. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 81–90, New York, NY, USA, 2007. ACM.

[12] William Jones, Harry Bruce, and Susan Dumais. Once found, what then?: A study of "keeping" behaviors in the personal use of web information. In *Proc. of ASIST*, 2002.

[13] Hyuckchul Jung, James Allen, Nathanael Chambers, Lucian Galescu, Mary Swift, and William Taysom. One-shot procedure learning from instruction and observation. In *Proc. of FLAIRS: Special Track on Natural Language and Knowledge Representation*, 2006.

[14] Jing Liu, Xin Dong, and Alon Y. Halevy. Answering structured queries on unstructured data. In *Proc. of WebDB*, 2006.

[15] Jayant Madhavan, Shirley Cohen, Xin Luna Dong, Alon Y. Halevy, Shawn R. Jeffery, David Ko, and Cong Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.

[16] R. Miller and B. Myers. Creating dynamic world wide web pages by demonstration, 1997.

[17] Planzo. http://www.planzo.com.

[18] Alex Safonov. Web macros by example: users managing the www of applications. In *SIGCHI Extended Abstracts*, pages 71–72, 1999.

[19] m.c. schraefel, Yuxiang Zhu, David Modjeska, Daniel Wigdor, and Shengdong Zhao. Hunter gatherer: interaction support for the creation and management of within-web-page collections. In *In Proc. of WWW*, pages 172–181, 2002.

[20] Abigail J. Sellen, Rachel Murphy, and Kate L. Shaw. How knowledge workers use the web. In *Proc. of the SIGCHI*, pages 227–234, 2002.

[21] Jeffrey Stylos, Brad A. Myers, and Andrew Faulring. Citrine: providing intelligent copy-and-paste. In *UIST '04: Proc. of the 17th annual ACM symposium on User interface software and technology*, pages 185–188, New York, NY, USA, 2004. ACM Press.

[22] Atsushi Sugiura and Yoshiyuki Koseki. Internet scrapbook: automating web browsing tasks by demonstration. In *Proc. of UIST*, pages 9–18, 1998.