Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

Proposal for Thesis Research in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy

Title: Using Semantic Web Technologies to Improve the Web Experience Today

Submitted by: David F. Huynh
905 Main St. #2
Cambridge, MA 02139

Signature of Author: .....

Date of Submission: November 15, 2005

Expected Date of Completion: June 2007

Laboratory where thesis will be done: Computer Science and Artificial Intelligence Laboratory

Brief Statement of Problem:

The World Wide Web's data model based on Web pages and Web sites is restricting the way Web users can mix and match and reuse Web content from multiple sites. By building Semantic Web technologies right into the Web browser, such limitations can be overcome, leading to more flexible and effective use of Web information, particularly for decision making. This approach also provides a gentle, users' needs-sensitive migration path from the Web to the Semantic Web.

Supervision Agreement:

The program outlined in this proposal is adequate for a doctoral thesis. The supplies and facilities required are available, and I am willing to supervise the research and evaluate the thesis report.

.....
David R. Karger, Assoc. Prof., EECS

.....
Robert. C. Miller, Asst. Prof., EECS

.....
Eric Miller, Semantic Web Activity Lead, W3C

Using Semantic Web Technologies to Improve the Web Experience Today

David F. Huynh
November 2005

Introduction

For all the riches of the World Wide Web, gathering resources to accomplish a complex information-centric task is not trivial. Resources supporting the task may be scattered across many corners of the Web. Although one can use search engines to reach most of those corners, it remains challenging at best and impossible at worst to bring back only the relevant bits, pool them together, perform analysis on the aggregate, and visualize the data in ways that support the task at hand. Consider the following scenario.

ꕥ

Alice, Ben, and their children are moving to Boston and they are looking to buy a house. Being first time buyers, the couple do not know what to look for and the whole decision is quite daunting to them. Before they even meet with a realtor agent, they would like to better understand the issue of buying a house—what factors they should consider, how each factor matters to them, how much they should pay for a particular amenity, etc.

There are many resources for U.S. house buyers on the Web, although all seem to get their data from Multiple Listing Service (MLS). Different real estate Web sites provide different but similar browsing and search interfaces on top of the same data. As an example, the search interface offered by Realtor.com is shown on the next page (Figure 1).

Although this search interface is rich, Alice and Ben do not know exactly what they want to search for. Some of the features listed are obviously crucial to them (in particular, single-family home, under $700,000, 4 bedrooms, 2 bathrooms, 2 garages, dining room, family room, den/office, fireplace) while the others are certainly nice to have (e.g., swimming pool, tennis courts, mountain view). By specifying the must-have features, the couple refine the 3,700 plus listings in Boston down to 128. Each of the 13 results pages displays at most 10 listings in a list (Figure 2), each listing as a summary view with a “View details” link to a separate detail page (Figure 3).

The 128 listings differ in the features that Alice and Ben consider nice to have but not absolutely crucial. In order for the couple to understand their differences, and which of those differences matter to them, they need to obtain an overview of all these listings, arriving at big-picture observations such as that 20% of these houses have central air and 80% of those were built within the last 10 years. Unfortunately, they cannot get this overview by reading individual detail pages one at a time as supported by the Web site.

Another useful overview is a street map of these 128 houses. On such a map, Alice and Ben can more easily tell how close a particular house is to their offices, how far it is from highway exits. Unfortunately, no realtor site provides a cartographic view of their data. In fact, some realtor sites do not even provide the exact addresses of listed houses.

On this hypothetical map, Alice and Ben want to include other information such as the locations of bus stops, subway stations, grocery stores, and restaurants. They also want to add wireless-enabled coffeeshops (where Alice, being a writer, loves to hang out while conduct-

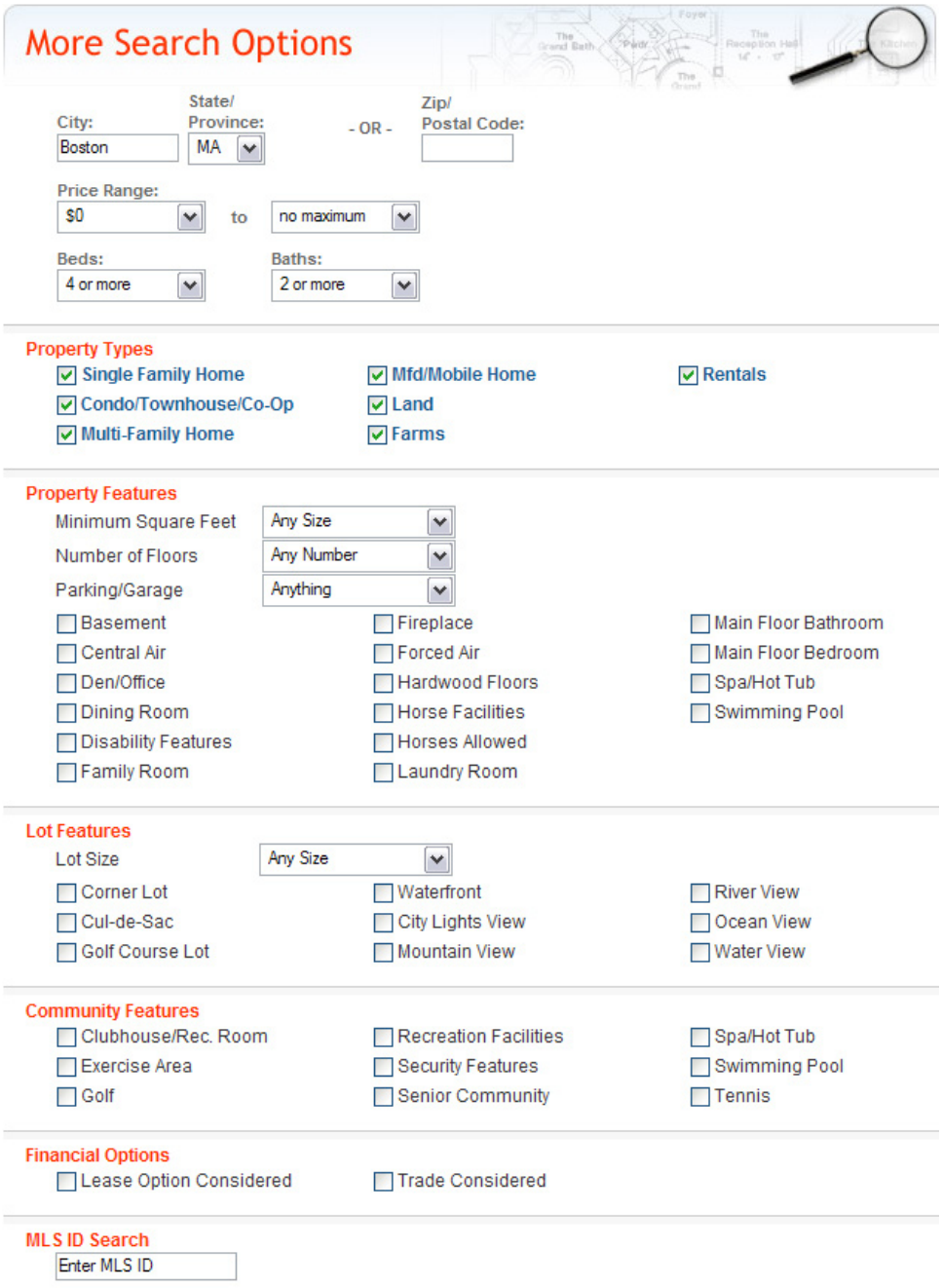

Figure 1. Realtor.com search options are plenty, but not helpful for someone who does not know what to look for in a house

ing character research for her novels) and arts supplies stores (where Ben gets materials for his hobbies). They want to locate elementary schools and Sunday language schools for their children. They want to stay clear of crime hot-spots but close to hospitals (as their children are often ill).

The realtor Web sites do not offer such information. Alice and Ben must find the extra information on other Web sites. These other Web sites might not provide map views of their information. Even if they all do, Alice and Ben want to see all the information on the same map rather than on different maps at different sites.

In addition to being overwhelming, simply seeing all such relevant information on a single map is not enough. Alice and Ben want to "play" with the information, e.g., filtering out elementary schools with bad public opinion ratings, selecting only restaurants offering their favorite cuisines, picking hospitals by specialization. As they tinker with the data, the map updates to show the combination of their current choices. In this manner, they come to discover the trade-offs they will have to make. For example, if they want to live near Asian restaurants, they will be far from hospitals with high-quality child care units.

Figure 2. Search results show few details per item, and the mapping feature does not work

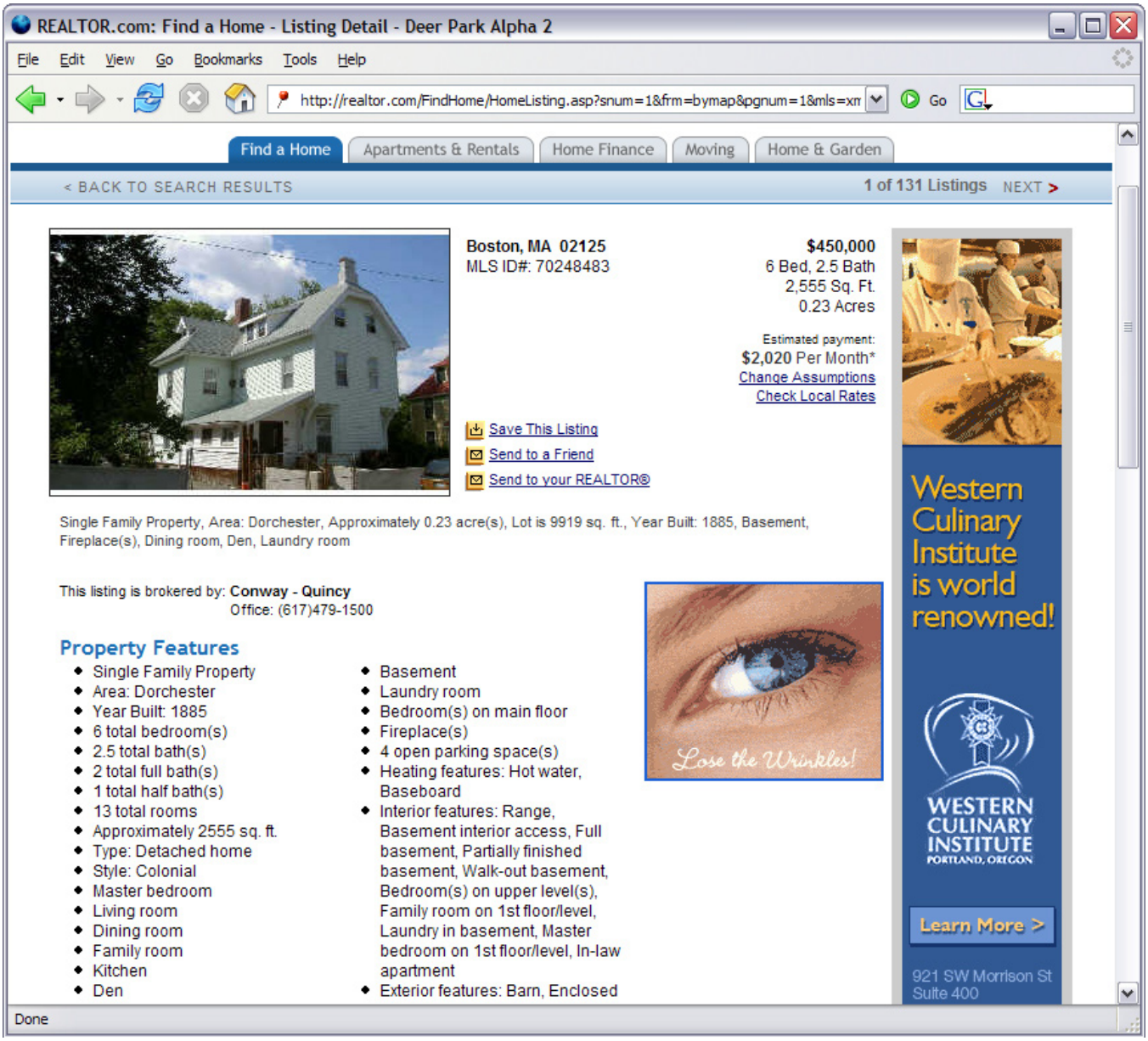

Figure 3. Details page for a house on sale

Having selected a few candidates from all the houses for sale, Alice and Ben check out their neighborhoods through street-level drive-by photographs, such as those offered by the A9 Maps service [1]. Through these photos, they obtain a more candid perspective of these houses than through the carefully staged photos on the realtor sites.

One of Alice's and Ben's close friends, Charlotte, has recently bought a house in Boston. Alice and Ben would like to understand Charlotte's choices and her decision. Unfortunately, Charlotte's research data is scattered in text files and spreadsheets. It cannot be incorporated into Alice's and Ben's hypothetical map to guide their own research.

Charlotte does give one tip: for each house, take into account the costs of immediate repairs and large appliance purchases. In order to perform such computations, Alice and Ben must first create a spreadsheet and copy data from the realtor Web site into it. They must then keep it up to date with the sources.

This research process goes on for several weeks. It is not conducted on a single computer: whenever Ben is waiting at airports, he searches for large appliances to buy, and whenever Alice finds free time at work, she reads through online home owners' advices and rates each candidate house according to these advices. Their research findings are gathered together at their home computer, and gradually, they come to understand the information space of house buying, or at least the aspects that matter to them.

ꕥ•ꕥ

The difficulties that Alice and Ben face in their house buying research highlight some limitations of the current Web:

- Users are not in total control of how Web information is
  - browsed,
  - viewed, and
  - computed.
- Information from several Web sites and several Web users cannot be easily mixed so that it can be used together.

Overcoming these limitations to help users benefit more from information on the World Wide Web is the motivation for my thesis.

Approach

I am proposing to create software that augments the user's Web experience through the following capabilities:

- Collection: Bits of Web information can be re-encoded in the Semantic Web data format (the Resource Description Framework, RDF [16]) and stored locally permanently. Re-encoding can be done by installable custom scripts or through by-example UI actions.

- Augmentation: Re-encoded data is used to augment the original Web pages where the data comes from, allowing for the support of faceted browsing from within the Web pages.

- Integration: Re-encoded information collected from different Web sources can be mixed together locally, letting the user pool together information tidbits that are related regardless of where they come from.

- Browsing: Re-encoded data can be browsed through a faceted browsing UI [43] adapted for RDF data. While classical faceted browsing techniques deal with one set of first-class items sharing multiple properties, this adaptation deals with many sets of first-class items that are interrelated. One set of items can be used as a facet of another.

- Visualizing: Re-encoded data can be re-visualized on tables, maps, calendars, timelines, and simple plots.

- Computing: Computations can be performed on the re-encoded data, e.g., sum, average, subtotal, pivot tables, string manipulations, in order to help the tasks of visualizing and browsing.

I intend to study the usefulness of this set of capabilities in decision making based on information on the Web.

My thesis statement will be

> **Embedding Semantic Web technologies inside Web browsers and applying them on existing Web information allows for more effective use of the Web, particularly for personal decision making.**

I am aiming to make the following contributions:

System

- An RDF-based blackboard architecture for Web screen scrapers to pool together their data gathering efforts.

User Interface Designs

- A nested faceted browsing UI that adapts faceted browsing principles to work on deep RDF graphs, with implication on how browsing the Semantic Web might work in the future.

- An adaptation of the spreadsheet UI for performing computations and constructing visualizations on RDF data in an object oriented fashion.

User Interface Techniques

- A technique called *link sliding* for constructing nested queries by clicking on joint predicates.

- A technique called *divide & conquer* that lets subcollections be refined separately while the whole collection is viewed together.

Study Conclusions

- The abilities to customize the ways Web information is computed, browsed, and viewed and to merge Web information from several sources let users make decisions based on such information faster, more easily and with greater satisfaction.

I also aim to make a non-academic contribution through this claim:

- Leveraging Semantic Web technologies as the means to address users' existing needs on the Web will drive the users' adoption of the Semantic Web.

User Interface

This section describes the preliminary user experience design of the software I intend to create. It first provides an overview of the design and then focuses on individual functionalities as mentioned in the Approach section.

Overview

The software in question is designed to augment the user's experience in browsing the World Wide Web. This requirement translates to two design philosophies:

- The software does not intrude except when called upon.
- It follows the Web's and the Web browser's interaction model as much as possible.

So that it can be ready to help out as the user browses the Web, and so that it can provide a smooth, integrated experience together with the Web browser, the software is to take the form of a Web browser extension. It will hereafter be called the *Piggy Bank* extension, or just Piggy Bank.

In each browser window or browser tab, Piggy Bank surrounds the content area with its own UI elements. Figure 4 shows the generic elements that Piggy Bank adds. These elements differ in different situations. They are hidden by default and then revealed when Piggy Bank's help is needed.

Piggy Bank also adds to the Web browser various menu commands and keyboard shortcuts as needed for invoking its functionalities.

Collection

The first functionality offered by Piggy Bank is the collection of Semantic Web data from Web pages. For each Web page, there are 3 ways for Piggy Bank to get this data:

- The data is already provided by the Web page's author, either through the <link> tags in the page, or as RDF-A [15] elements inside the page, or recoverable by applying GRDDL [5] transformations (linked from the page) on of the page itself.
- There are third-party *screen scrapers* that can re-encode data within the page into Semantic Web format. Piggy Bank can run these screen scrapers on top of the page. (For security reasons, the user must install and activate these screen scrapers first.)
- The user herself can mark up patterns of the data on the page and let Piggy Bank follow those patterns to extract out the data.

Figure 5 shows the sidebar of Piggy Bank as the user proceeds to collect data from a Web page. First, all available means for getting the data is presented so that the user can choose. Preset preferences can be applied on these choices automatically and by default, the user only has to click "Auto Collect". The collection then proceeds with visual feedbacks. Once the data has been collected, the sidebar shows browsing controls which will be discussed in a later section. The original Web page remains but gets augmented, as will also be discussed subsequently.

Note that some parts of the data collected might not be immediately available on the original Web page. For example, if the original Web page is a search results page, some parts of the data collected might come from subsequent search results pages. Piggy Bank leaves to

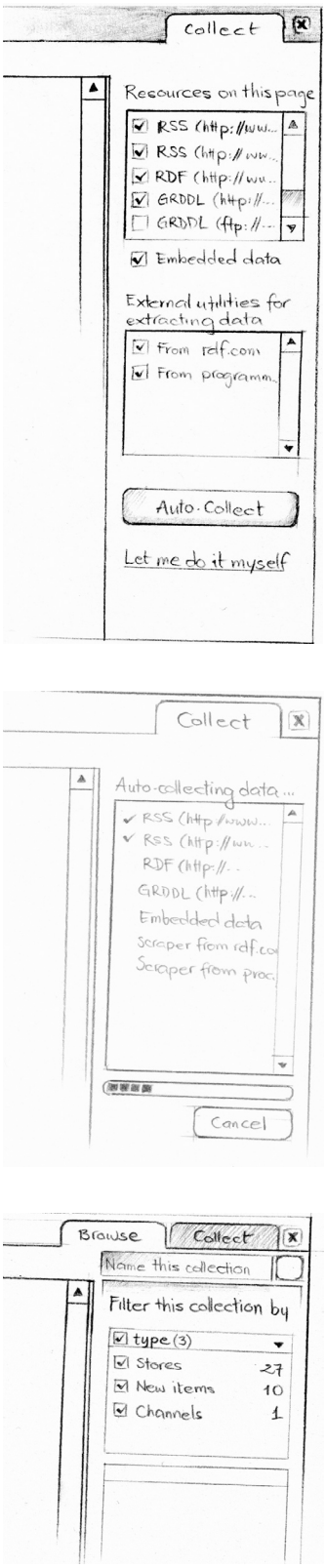

Figure 5. Collecting data from a web page

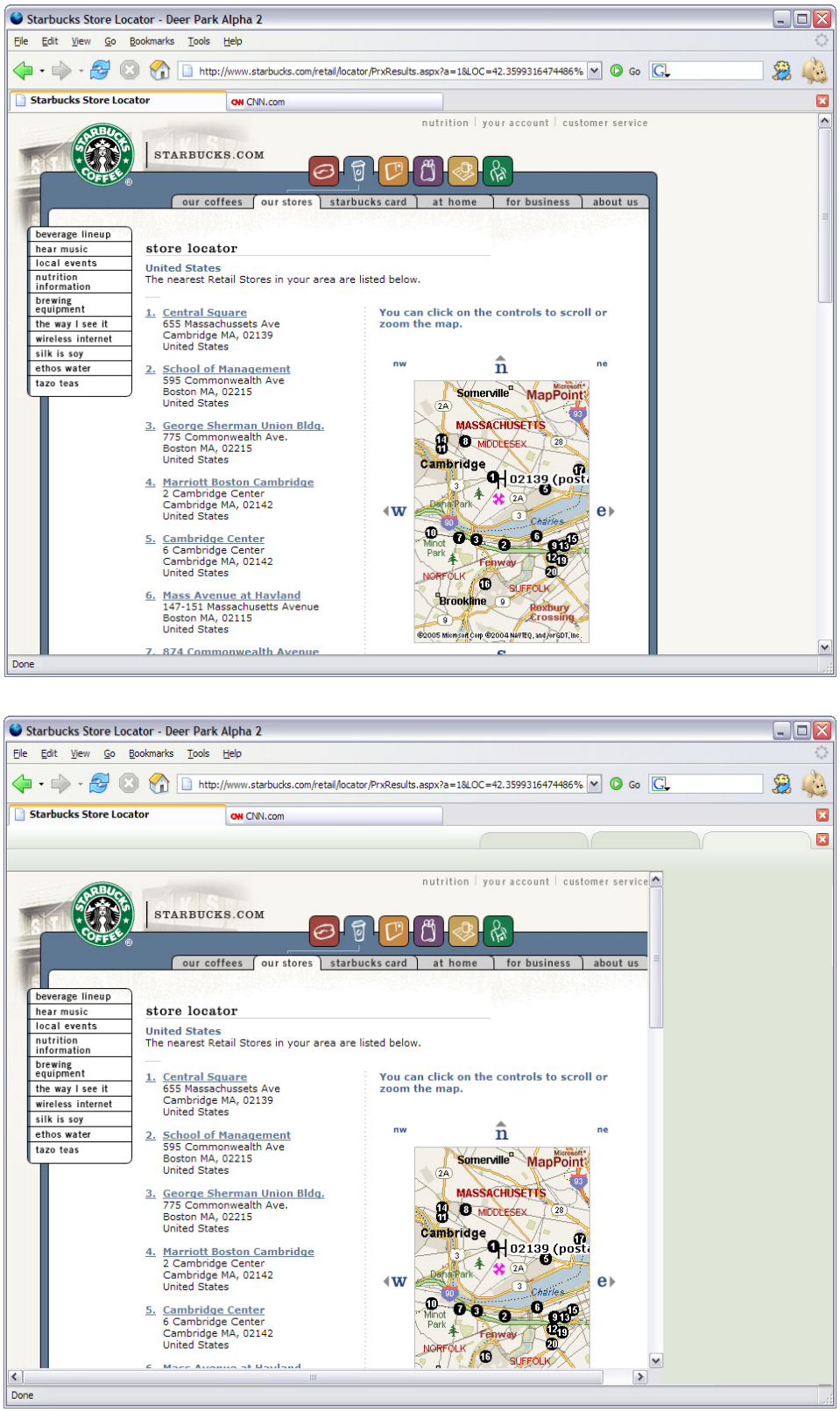

Figure 4. User interface elements added to each Web browser window or tab by Piggy Bank

the discretion of the Web page's author and the screen scrapers' writers as to what data can be collected from the Web page. Presumably, this data will correspond to the user's mental model of what data is contained, presented in, or communicated through the page.

As mentioned previously, the user can also take matter into her hands and scrape the page herself. However, the user interface for this task is not yet designed.

The intended experience is that having invoked Piggy Bank on a Web page, the user suddenly has more means to interact with the data therein contained but without having to shift to a different user interface or a different application.

Augmentation

Figure 6 shows how Piggy Bank might augment the first search result on the Web page shown in Figure 4. A border is drawn around the search result and clickable icons are attached to it. The bottom icon shows a context menu corresponding to the nature of the item represented by the augmented element (cf. Microsoft's Smart Tags). In this case, the augmented element represents a store, which can be located on a map and has a phone number that can be called. The context menu also contains browsing commands not offered by the original Web page. For example, using this context menu, the user can refine all the search results down to just 2 Starbucks stores located on Massachusetts Ave. Clicking on the top icon shows the Tags bar where the user can assign keywords to the item. These browsing and tagging functionalities will be discussed subsequently.

When the user returns to a Web page from which data has recently been collected, Piggy Bank will immediately show browsing controls and augment that page. This also applies to pages whose data has been collected as the result of some other pages being collected. For example, the user can let Piggy Bank collect data from just one search results page but also automatically gets Piggy Bank's help on subsequent search results pages.

Integration

There are 3 mechanisms for integrating data collected from different sources:

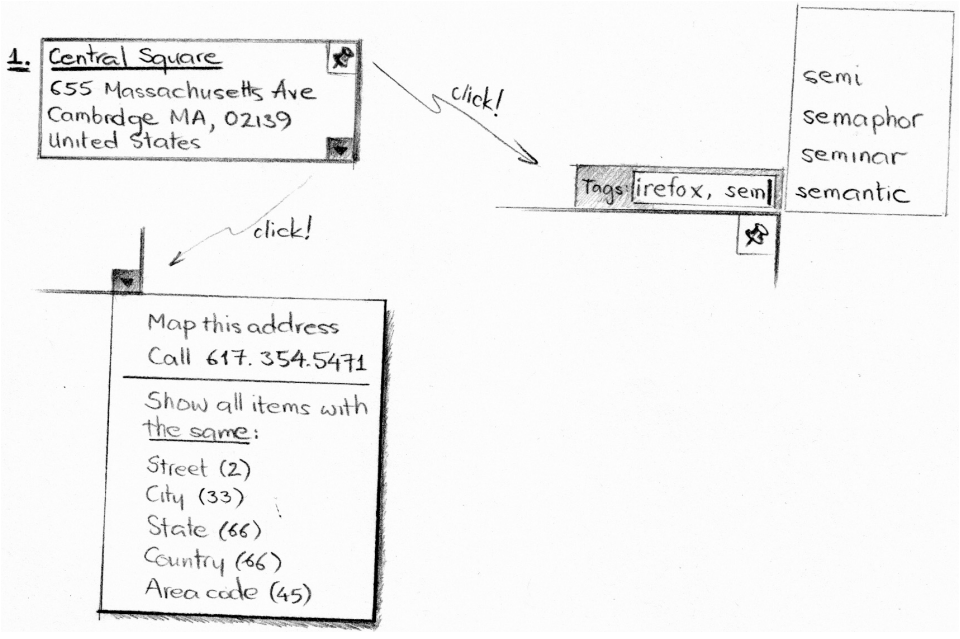

Figure 6. Augmented element on a Web page

Saving

Collected data items can be saved locally into a common repository named "My Piggy Bank". This repository can be browsed just like a Web site inside the Web browser. But unlike for a Web page, Piggy Bank's browsing controls are automatically provided—no collection phase is required since the data is already in Semantic Web format.

Tagging

Each collected data item can be tagged with one or more keywords. Once tagged, it is also saved automatically. Keywords can then be used to query for items similarly tagged regardless of their origins.

Mixing

Set operations (union, intersection, exclusion) can be performed on collections of data items being browsed. When collections are mixed, a new repository is created to hold the resulting collection, which can then be browsed just as the "My Piggy Bank" repository can be.

Browsing

A consequence of RDF's lack of schema enforcement and its URI naming scheme is that information items encoded in RDF can be easily "elaborated" as more and more is known about them. For example, the user might get a book review from one Web site which provides only the name of the book's author. When the user visits the author's home page, she is able to gather the author's contact information (e.g., office address and phone number) as well as discover which schools he has lectured at. Then by visiting those schools' Web sites, she learns of the courses they offer that are relevant to the book's topic. This process of *data elaboration* both

- *widens* the RDF graph
  - by adding more properties on invididual information items (e.g., office addresses and phone numbers on authors) and
  - by adding more item types that share some properties (e.g., both schools and authors have addresses); and
- *deepens* the RDF graph by yielding a mixture of information items of different types (books, people, schools, courses) that are inter-related.

In a wide RDF graph, a collection of items might share several properties by which the user can use to refine to only a desired subcollection. For example, the office addresses of several book authors can be used to narrow down to only those authors who live within some distance from the user. Then these authors' native languages can let the user determine whom she can talk to directly and whom she needs an interpreter for. Such browsing needs based on multiple shared properties are currently best supported by the faceted browsing paradigm [43].

However, so far faceted browsers only support homogeneous collections, each containing items of a common type (e.g., Flamenco on fine arts and architectural images [4], Epicurious on recipes [3]). They do not operate on heterogeneous collections such as one which can be composed of authors and schools. The user might wish to maintain such a collection while refining it further based on the authors' native languages and the schools' majors. Applying a restriction on native language on the entire collection will eliminate all schools. To address this need, a new functionality called *Divide and Conquer* will be introduced in a later subsection.

The faceted browsing paradigm also does not support *deep* corpora, those containing items of different types that are inter-related. First, there is a need to shift focus from one collection of items to a related collection of items, e.g., browsing from books to their authors, then from the authors to their schools. This action is just like hopping from one Web page to another, except that it operates on collections, not individual items. To address this need, a new browsing mechanism called *Link Sliding*, or just *Sliding*, will be introduced in a later subsection.

One collection of items can also be refined based on its related collections of items. For example, the user might wish to narrow down a list of authors not just by their addresses but also by the addresses of the schools they have lectured at or by the locations of the conferences they have attended. The current faceted browsing paradigm must be adapted to support this need.

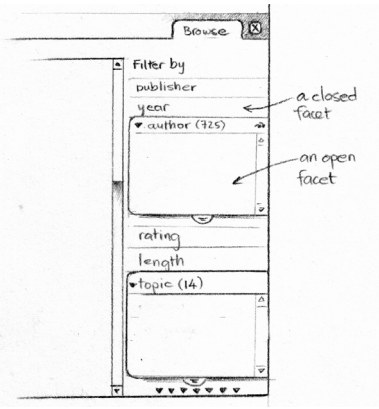

Figure 7. Facets

The overall browsing paradigm consists of 2 kinds of action: filtering the currently browsed collection of items to arrive at a desired subcollection, and sliding from one collection to another.

Filtering

Figure 7 shows several facets on the sidebar of Piggy Bank as the user browses a collection of items. Filtering is performed by selecting values shown within these facets. Figure 8a indicates that the user has filtered a collection of books to only those written by Margaret Alida. The number beside each value is the preview of the number of items resulting by checking that value. More than one value per facet can be checked. If no value is checked in a facet, there is no restriction on that facet.

Figure 8b shows the dropdown menu of a facet. It allows the user to perform a negation (i.e., all except the selected values). The user can also group the values by their own properties, such as grouping the authors by the schools they have lectured at (Figure 8c). The user can continue to group those groups further (Figure 8d) until the number of values is small enough or the grouping makes enough sense to perform selection.

Facets need not contain a list of values. They can take other forms, such as a calendar (Figure 9), a timeline, a map, a plot.

Sliding

Link hopping is traversing from one item along one link to another item. Link hopping is the fundamental mechanism for surfing the Web. Link sliding is traversing from a collection of items along several links sharing some common characteristics to another collection of items. Now that on the Semantic Web, links are named by URIs and their characteristics

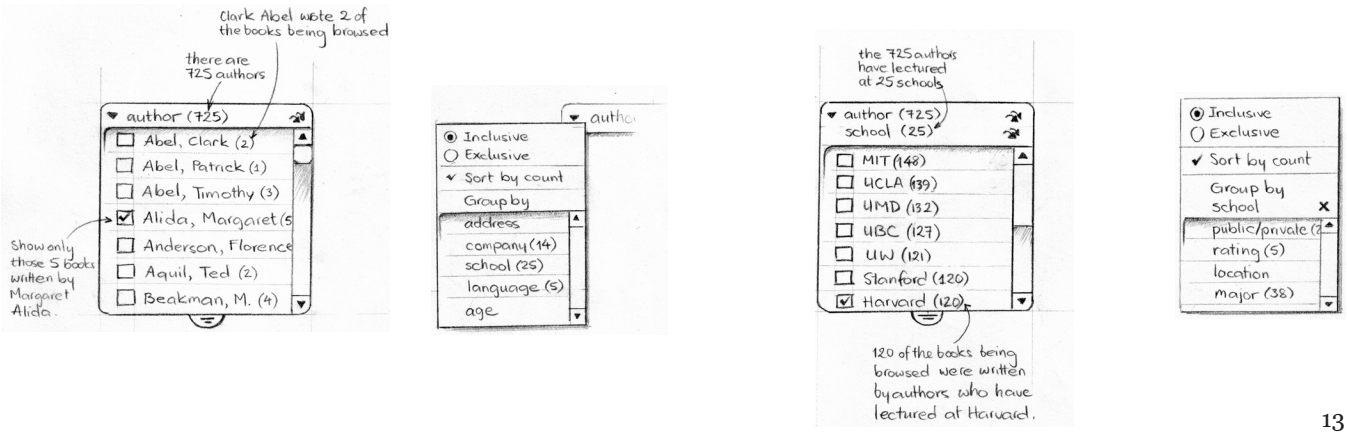

Figure 8. Inside a facet: a) values show count preview; b) dropdown menu gives grouping options; c) grouped values show count preview; d) dropdown menu gives further grouping options

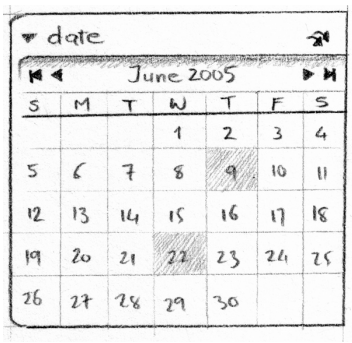

Figure 9. A date facet

can be recorded, I posit that link sliding will be the fundamental mechanism for surfing the Semantic Web and corpora of Semantic Web information.

In Piggy Bank, there are 2 modes of link sliding:

- forward sliding: the destination collection is a subset of the entire corpus; forward link sliding will be used pretty much like link hopping to change focus from one collection to a different, related collection; and
- backward sliding: the destination collection is a subset of some other collection previously browsed on the path to the current collection; backward link sliding will be used to filter a collection by sliding forward, filtering, and then sliding backward.

Figure 10 illustrates the difference. The user has arrived at a collection of 3 items from another collection of 5 items. The user then filters the 3 items down to only 2. Sliding forward along the links shown yields 4 items, including 1 that has not been seen before. Sliding backward, on the other hand, yields only 3 items that belong to the previously browsed collection of 5 items.

Note that "forward" and "backward" do not concern the direction of the RDF predicates along which link sliding occurs. "Forward" and "backward" only determine whether the resulting collection will be taken out of a previously browsed collection or out of the whole corpus.

Forward sliding can be performed by clicking on the double arrow icon on each facet. For example, in Figure 8b, clicking on the double arrow icon slides forward to the collection of 725 authors.

Backward sliding can be performed by clicking on the various segments of the trail as shown in Figure 11. The directions of the arrows on the trail are the directions of the RDF predicates. The trail thus reads: [1579 books] written by [725 authors] who have lectured at [25

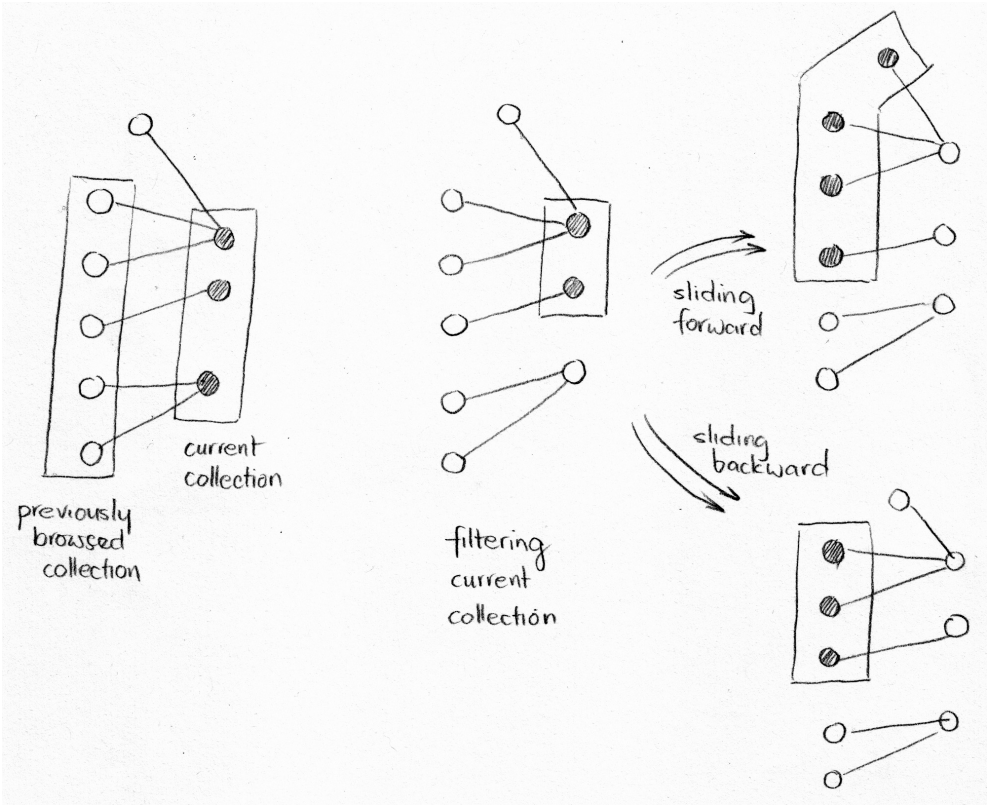

Figure 10. Sliding backward and forward

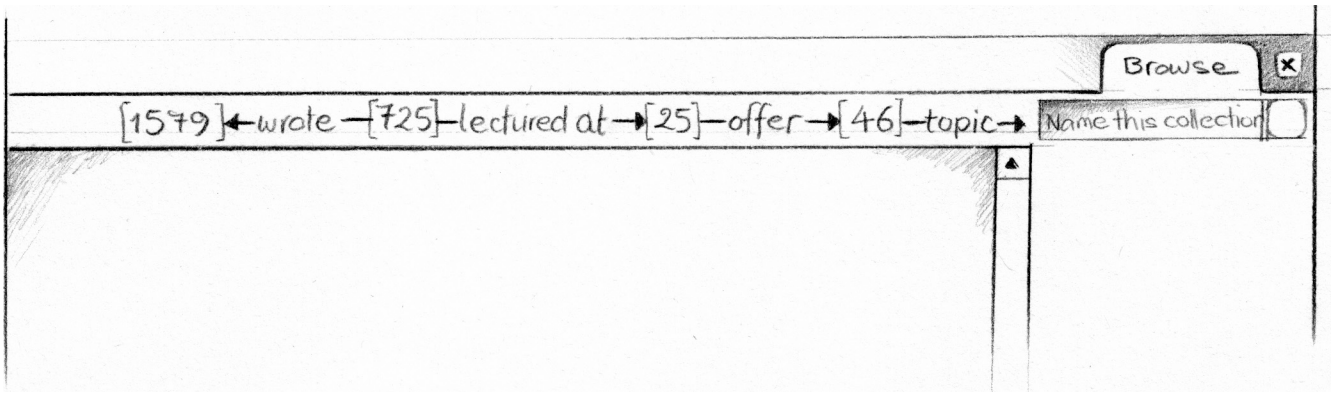

Figure 11. Browsing trail for sliding backward

schools] which offer [46 courses] on the [topics] being browsed. As the current collection of topics is filtered, the counts on the trail are updated.

Divide and Conquer

Filtering and sliding let the user moves from one collection to a subcollection or to a related collection. They are designed to deal with one collection at a time. There are occasions where multiple conceptually different collections need to be handled together. For example, the user might wish to plot on the same map houses on sale, restaurants, hospitals, schools, etc., and to filter these collections down individually while seeing the aggregate view updated on the fly. The user might choose to see only restaurants serving a particular cuisine, hospitals with a particular department, schools of a particular rating or higher, etc. Each of those filtering criteria applies only to one collection, not the whole aggregate view.

Even if the filtering criteria can apply to all collections being browsed, the user might still wish to apply them to only one collection. For example, consider browsing a collection houses on sale. For houses in different neighborhoods, you might have different sets of criteria: in good neighborhoods you focus on prices but in bad neighborhoods you place more importance on amenities.

The Divide and Conquer feature lets the user apply filtering criteria and sliding actions to only one subcollection of the currently browsed collection. The user first chooses a criterion by which to split the collection into several subcollections. This can be done by picking a facet, performing appropriate groupings, and then invoking the Divide and Conquer command on its dropdown menu (not shown in preceding renderings). The collection will be split by the values in that facet. Then that facet is removed from the sidebar and a dropdown list will be shown above all the facets, letting the user select which subcollection to focus on.

Landmarks

Piggy Bank's browsing controls are designed to encourage the user to tinker with the information interactively through lightweight actions and quick feedbacks. This interaction model departs from the traditional "whole Web page" interaction model and follows the trend of recent *AJAX* [24] Web applications, approaching desktop applications' level of interactivity.

However, the traditional "whole Web page" interaction model has one advantage that should not be lost: it allows the user to capture the state of the application by bookmarking the current Web page, so that this state can be easily restored in the future. States can also be easily restored using the browser's history and Forward and Back buttons.

So that the many interactions through Piggy Bank don't overwhelm the browser's history, such that the Back and Forward buttons still prove useful, only some of the actions will get logged into the browser's history. They are: sliding actions, divide-and-conquer actions, and

view changing actions (which will be discussed later). These actions change what the user is conceptually looking at. Filtering actions, on the other hand, are more for experimenting with the current collection; they are numerous and they don't switch the current collection.

The pages browsed through Piggy Bank that are logged into the browser's history are called landmarks. There is a tab in Piggy Bank's user interface that explains how the user has gotten to the current landmark.

Visualizing

Piggy Bank will offer several basic ways to visualize collections of items.

Original Web Page

When a collection of items has been collected from one or more Web pages (e.g., search results pages from a single search), then those Web pages, augmented, are offered as a view for that collection. This is to keep the user's context right after information has been collected from those Web pages. Actions that filter the collection or slide away from it will switch to another view.

List

Items are possibly grouped, then sorted by some order, and paginated into one or more pages. The user can control all 3 steps. Each group offers UI for filtering the collection to just that group.

Table

The user can construct a table to view a collection of items. Each item occupies one row. Columns can be added to show the items' properties. Items can also be grouped and sorted.

Plot

Simple plots can be made from tables just as in spreadsheet software.

Map

Items with geographical coordinates (direct or indirect) can be placed on a map. The user can use their properties (direct or indirect) to format the markers (e.g., specifying colors, shapes) on the map. Rectangular selection bands and zooming controls let the user filter the collection.

Calendar and Timeline

Items with times and dates (direct or indirect) can be shown on a calendar or a timeline. Their properties can also be used to format the calendar and the timeline. Custom controls are offered for filtering.

Computing

A simple object-based language will be offered to let the user perform computations. For example, in the table view of a collection, the user can add a column and specify the data formula for it as "item.price * 1.15" in order to compute the 15% tax-included price of each item. Then at the bottom of that column, the user can specify a summary formula, "sum(column.dataset)", to find the total expenditure. The user can also specify the format for each cell of

that column with the formula, “cell.style.backgroundColor = if(item.price < 40, ‘yellow’, ‘white’)”.

Evaluation

I will test the usability of Piggy Bank's user interface, particularly its faceted browsing functionalities. This is carried out by having user study subjects perform browsing operations to refine a collection of data items to some particular subsets. Several different subsets will be requested, each designed to elicit the use of a particular functionality in the browsing interface (e.g., grouping, sliding, negation). Success rate will be recorded and subjective feedbacks will be collected.

I also intend to verify the benefits of Piggy Bank for decision making based on information on the Web. In particular,

1. Piggy Bank's features (augmentation, browsing, visualizing, computing) offered on re-encoded data from a Web site enable users to make more satisfying decisions based on Web information; and

2. Piggy Bank's ability to integrate information from several Web sites lets users make better use of those sites.

A user study will be conducted on two groups of subjects, one group given Piggy Bank to perform a decision making task and the other group given only a Web browser. Task completion time will be measured and the subjects will be ask to judge their satisfaction with their decisions.

A survey will be sent to Piggy Bank's users soliciting descriptions of the data they have collected. This is to determine whether they have had the need to integrate information from several Web sites and have been enabled to do so by Piggy Bank.

Related Work

The World Wide Web has liberated information from its physical containers—books, journals, magazines, newspapers, etc. No longer physically bound, information can flow faster and more independently, leading to tremendous progress in information usage.

But just as the earliest automobiles resembled horse carriages, reflecting outdated assumptions about the way they would be used, information resources on the Web still resemble their physical predecessors. Information is presented in units called Web pages, served from separate sources called Web sites. This information model—based on Web pages and Web sites—limits how users make use of the Web as a whole.

Web directories and search engines were invented in effect to break down Web site barriers, letting users query the Web as a whole rather than as separate, independent parts. However, Web directories and search engines only lead users to Web pages. They do not help users repurpose the information within to some form that suit their needs. They do not let users mix information from different sites.

In this section, I first explore several types of approach that let users repurpose Web information and achieve utility over information from different sites, and then discuss previous work related to the functionalities described in the User Experience section.

Server-Side Approaches

Recognizing users' need to overview information from different Web sites, Web portals such as Yahoo! compose aggregate views of news article summaries, weather reports, stock quotes, etc. The composition rules for these views—sources of the information and layout templates—can be edited by users or generated using machine learning algorithms [38]. These portals offer only information in a few fixed domains and they are designed to provide up-to-date information rather than help users aggregate information relevant to specific needs at specific times. Furthermore, while they can collect information from different sources into a common view, they do not allow manipulations of that information into alternative forms that might facilitate more effective use (e.g., plotting news articles on a map or fitting them onto a calendar).

A more recent form of aggregating and re-presenting information makes use of Google Maps' Web APIs [6] to map information not offered in cartographic view by the original Web sites (e.g., crime scenes [2], apartments for rent [9]).

These server-side approaches to aggregating information simply move the decisions of packaging and presenting information from one publisher to another, still leaving users out of the process. While a user can now see crime scenes on a map and apartments for rent on another map, he or she needs yet another publisher to marry the two maps together as it might be desirable to rent apartments far from crime hot-spots. The approach whereby a Web site aggregates and re-presents information from some other Web sites does not scale to suit the personal needs of all Web users.

Client-Side Approaches

Users can get custom news front pages at Web portals or they can employ screen scrapers on favorite news Web sites on their local computers. In the past, only those with programming skills could use screen scrapers. Recent tools like Greasemonkey [8] and Chickenfoot [22]

are making Web page scraping a little easier. LAPIS [30] supports structured editing of Web pages and allows users to repurpose information on one Web page at a time. These tools are generally used for modifying Web pages rather than collecting information from them or aggregating information from several Web sites. Each of them lacks a rich and common data model necessary to pool and persist diverse information together.

Users can bookmark fragments of Web pages from different Web sites and manage them together using tools such as Hunter-Gatherer [37], NetSnippets [12], and Onfolio [13]. However, these tools do not recover the original structures of the bookmarked information and hence, they cannot offer functionalities based on such structures including support for faceted browsing, re-visualization, and custom computations.

Fixing the Web's Information Modeling

Standards such as XHTML, XML, XSLT, and CSS aim to separate Web data from presentational elements and attributes, easing the electronic publishing process. Although they can make Web documents cleaner and thus more conducive to be repurposed by users, there is no demonstration of this benefit in general except for creating personal news portals on the client-side by aggregating RSS feeds.

In fact, the hierarchical nature of XML will limit the ways in which XML data can be reused. An XML element (and its descendent elements) cannot be extracted out of the containing XML document without losing its identity as encoded implicitly by its ancestor elements and the XML document's identity. Perhaps for this reason that when XML is used for mixing content from different sources, i.e., aggregating RSS feeds, it is used only to mix lists of things rather than trees of things.

In order to allow information to flow freely, we must be able to describe information without wrapping containers around it. Enter the Semantic Web [18].

The Semantic Web

In the Semantic Web's information model (called the Resource Description Framework (RDF) [16]), every data item can be named universally by a Unique Resource Identifier (URI). As such, each item can be talked about anywhere rather than just within the context (read, containing document hierarchy) where its data was originally served. It can be extracted out from whatever packaging that contains it without losing its identity.

Despite its appealing promise to make the Web better, building the Semantic Web has been a decade old, on-going mountain-moving effort that has yet to benefit mainstream Web users. Without sufficient Semantic Web infrastructure in place, it is difficult to demonstrate to Web users the values of the Semantic Web. The agents touted to take advantage of rich Semantic Web meanings embedded within the Web to automate tasks on behalf of users have little Semantic Web data to run on. Without understanding the benefits of the Semantic Web, users show little demand for it. Without user demands, Web publishers find little incentive to offer Semantic Web data. And so the vicious cycle revolves.

Perhaps what makes the Semantic Web so good for users is what makes it so hard to accept for publishers. That idea of freely flowing information, easily repurposable and redistributable by users, is unpalatable to publishers who want to control their own information because their business models depend on such restricted access. Note that, while the Semantic Web was designed to benefit users, XML was designed to benefit publishers. XML allows for exchange of data in rigid schemas; its hierarchical nature facilitates a form of publisher-controlled packaging. In this way, XML makes it easier for publishers to exchange data in the controlled ways that they want. Publishers, with their abundant resources and economic

incentives, quickly adopted this technology that immediately benefited them. On the other hand, they might see the Semantic Web as a threat to their business.

Semantic Web in the Small

The Haystack platform [27] was a notable effort to bring Semantic Web technologies to users, in hope to leverage these technologies for personal information management and to help build the Semantic Web. Huynh, et. al., the system's creators, claimed that the use of a semistructure data model—that of the Semantic Web—would facilitate the construction of more flexible personal information environments. Semantic Web technologies were used to break down application barriers, allowing data of different types to mix, yielding more synergies than when they were kept apart in incompatible formats and different user interfaces. Haystack and other systems (e.g., Gnowsis [36]) now form a new type of infrastructure, the Semantic Desktop [17].

Unfortunately, by focusing on desktop personal information management, these semantic desktops have to challenge a plethora of commercial applications in this space that have fiercely competed and evolved over more than two decades to support tasks in a few limited information domains (e.g., e-mail, contacts, calendar, multimedia). These applications have been highly optimized for these tasks, and whatever new functionalities that the semantic desktops offer appear exotic and unreal in everyday's use.

Furthermore, Semantic Web technologies are designed for the scale of the Web and using them in a few familiar information domains is like using a nuclear reactor to power a family minivan—much research is needed to tame its power before it can prove both useful and safe for mainstream adoption. For example, while claiming much about the synergies from mixing semistructure data of every imaginable type, Haystack can only show limited, preliminary evidence supporting the usefulness of mixing heterogeneous items in a collection [33], a functionality that can be implemented without Semantic Web technologies. It is too much to ask of a user to adopt an entirely different information environment that offers limited and unclear benefits together with potential damages to the information that she relies on daily.

Harvesting the Web

Hogue, et. al., offered a connection between Haystack and the Web through Thresher [26], a tool that lets users extract information within Web pages and store them into Haystack in Semantic Web format. They claimed that this approach reduced users' reliance on Web publishers to offer Semantic Web content and thus sped up the adoption of the Semantic Web. Unfortunately, the adoption of Haystack itself remains to be seen.

Hogue, et. al., focused only on the algorithms used to automate the extraction and pointed to Haystack's generic information management functionalities as the means for users to repurpose the harvested information. However, Haystack's functionalities have never been designed or tested for the task of repurposing Web information. It is here that I would like to make my contributions—to bring the riches of the Web and the power of the Semantic Web together in a useful form adoptable and usable by users.

ꕥ•ꕥ

Having laid out the general space surrounding my thesis problem, I now delve into the specific work related to the various functionalities that I intend to implement.

Collection

Bookmarking fragments of Web pages is supported by the research tool called Hunter-Gatherer [37] and commercial tools such as Netsnippets [12] and Onfolio [13]. As mentioned before, these tools cannot recover structures of the information within the Web pages.

On the other hand, Thresher [26] recovers structures and stores the extracted information in RDF. RSS readers, in a sense, let users collect structured information from within Web pages and mix them together to redefine news channels according to their own tastes.

Chickenfoot [22] and Greasemonkey [8] allow users to write scripts to manipulate Web pages in any imaginable way. However, so far they have remained mostly a programmer's tools and the scripts mostly make simple augmentations to individual Web pages. These tools lack a rich and common data model necessary for persisting and pooling information from different sources together.

Augmentation

Before the arrivals of Chickenfoot [22] and Greasemonkey [8], many browser extensions had already provided Web page augmenting functionalities such as highlighting search words and looking up definitions (e.g., Google Toolbar [7], WEBI [19]). Chickenfoot and Greasemonkey let users create, share, and install these kinds of augmentation more easily.

Annotea [28] lets users add annotations to fragments of Web pages and share them with one another. When a user browses to a Web page that has been annotated, the page is automatically augmented with the annotations.

Integration

Integration of Web information on the client-side so far is only supported in the form of bookmarking Web pages and Web page fragments. Organization is supported through folders and keyword categorization. Aggregate reports of findings can be generated by juxtaposing Web page fragments (see Netsnippets [12] and Onfolio [13]).

Browsing

The Magnet navigation framework [40] in Haystack supports faceted browsing on arbitrary RDF data. It makes use of schemas only when available, and automatically resorts to machine learning techniques applied on actual property values for suggesting the most suitable facets by which to refine a particular collection of data items. However, Magnet requires explicit schema annotations to help it choose which indirect properties (properties of properties) to apply machine learning techniques on, as it can be expensive to learn over all indirect properties. Magnet's user interface does not provide a convenient way for users to add such schema annotations. Overall, Magnet behaves much like a traditional faceted browser (e.g., [43]) but does not require explicit schemas and can work on heterogeneous collections. While addressing the "wide" nature of RDF, unlike Piggy Bank's browsing interface, Magnet does not specifically address the "deep" nature of RDF.

Depth of semistructured data has been addressed by DataGuides [25] as an adaptation of query by example [???]. The structure of a database is dynamically analyzed at runtime and shown as a tree. By specifying conditions on the nodes of the tree, the user can thus formulate a query. While addressing the "deep" nature of semistructured data, DataGuides remains mostly a query interface, not a browsing interface. DataGuides does not support dynamic queries [39] and query previews [23] that have proven effective for interactively exploring large databases.

The DOPE browser [42] provides an alternative rich browsing interface for large RDF repositories. It can also digest a Web site's structure and provide its own browsing support on top of that structure. However, it is designed mostly for exploring intersections between related concepts, which is not always the mode of exploration.

Visualizing

RDF data is usually visualized as graphs (e.g., IsaViz [10]) or trees and property-value tables (e.g., Protégé [14]). For a more complete list of RDF visualization tools, see [20]. Haystack departs from this "one view fits all" presentation strategy and lets the nature of each RDF resource, or a collection of resources, dictate its view [34]. Fresnel [21] strives to be RDF's standard presentation ontology, also built on view resolution and view description concepts. While capable, Haystack does not support maps, plots, and timelines, which are perhaps most useful for everyday's use and simple data analysis scenarios.

Snap-Together Visualization [31] and Visage [35] work only on relational and object-oriented databases but they provide a generic means for users to construct sophisticated views for information exploration. Spreadsheet programs such as Microsoft Excel provide tools for generic plot constructions. Other programs that let users construct custom scatter plots and bar charts only work in restricted domains (e.g., PhotoFinder [29] in the domain of photographs). The FOCUS table [41], although only explored as a tool for product comparison and selection, can be useful as a summary view for exploring structured data.

Computing

Spreadsheet programs are the most obvious generic end-user tools for performing computations. There have been efforts to add data models to spreadsheets (e.g., Improv [11], Model Master [32]) so to make computations more apparent, thus reducing errors. I will look to these pieces of work for techniques for specifying computations once models have been constructed.

Schedule

The following is a preliminary schedule:

	2005	
Mid	November	• Designed and paper tested nested faceted browsing UI.
End of	November	• Designed and paper tested visualizing and computing UI.
Mid	December	• Designed Piggy Bank 3.0's architecture.
	2006	
End of	March	• Completed major functionalities. • Submitted **UIST 2006** paper on Semantic Web browser extension architecture.
End of	June	• Cleaned up, stabilized, and optimized implementation, readying for version 3.0 **alpha release**. • Started on first draft of **thesis**.
End of	July	• Designed usability study.
End of	August	• Conducted usability study and analyzed results.
End of	September	• Built mechanism for getting survey responses on Piggy Bank usage • Readied for version 3.0 **beta release**. • Finished 30% of **thesis**.
Start of	November	• Submitted **WWW 2007** paper on augmenting Web pages with Semantic Web technologies (including usability study).
Mid	November	• Submitted **SIGMOD 2007** paper on nested faceted browsing UI (including usability study).
Mid	December	• Finished 60% of **thesis**.
		Possibly TA for UI class during Fall 2006
	2007	
End of	January	• Designed decision making user study.
End of	February	• Implemented additional features necessary for decision making user study. • Readied for version 3.0 **release**. • Finished 90% of **thesis**.
End of	March	• Conducted decision making user study conducted and analyzed results. • Collected survey responses.
Mid	April	• Finished first complete draft of thesis.
End of	April	• Submitted **ISWC 2007** paper on Semantic Web browsing paradigm.
		Take 2nd minor class during Spring 2007
Mid	September	• Submitted **CHI 2007** paper on decision making UI for Web information.

References

[1] A9.com Maps. http://maps.a9.com/.

[2] ChicagoCrime.org.

[3] Epicurious.com: the World's Greatest Recipe Collection. http://www.epicurious.com/.

[4] flamenco search home. http://bailando.sims.berkeley.edu/flamenco.html.

[5] Gleaning Resource Descriptions from Dialects of Languages (GRDLL). http://www.w3.org/2004/01/rdxh/spec.

[6] Google Maps. http://maps.google.com/.

[7] Google Toolbar. http://toolbar.google.com/.

[8] Greasemonkey. http://greasemonkey.mozdev.org/.

[9] HousingMaps.org.

[10] IsaViz. http://www.w3.org/2001/11/IsaViz/.

[11] Lotus Improv. http://en.wikipedia.org/wiki/Lotus_Improv.

[12] Net Snippets. http://www.netsnippets.com/.

[13] Onfolio. http://onfolio.com/.

[14] The Protégé Ontology Editor and Knowledge Acquisition System. http://protege.stanford.edu/.

[15] RDF/A Syntax. http://www.formsplayer.com/notes/rdf-a.html.

[16] Resource Description Framework (RDF) / W3C Semantic Web Activity. http://www.w3.org/RDF/.

[17] Semantic Desktop Workshop. http://www.semanticdesktop.org/.

[18] Semantic Web project. http://www.w3.org/2001/sw/.

[19] Barrett, R., P.P. Maglio, and D.C. Kellem. How to Personalize the Web. In Proc. SIGCHI 1997.

[20] Beckett, D. Dave Beckett's Resource Description Framework (RDF) Resource Guide. http://planetrdf.com/guide/.

[21] Bizer, C., R. Lee, and E. Pietriga. Fresnel — A Browser-Independent Presentation Vocabulary for RDF.

[22] Bolin, M., M. Webber, P. Rha, T. Wilson, and R. Miller. Automation and Customization of Rendered Web Pages. Submitted to UIST 2005.

[23] Doan, K., C. Plaisant, and B. Shneiderman. Query Previews in Networked Information Systems. In Proc. ADL, 1996.

[24] Garrett, J. Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/publications/eSssays/archives/000385.php.

[25] Goldman, R. and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In Proc. VLDB Conference, 1997.

[26] Hogue, A. and D. Karger. Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web. In Proc. WWW 2005.

[27] Huynh, D., D. Karger, and D. Quan. Haystack: A Platform for Creating, Organizing and Visualizing Information Using RDF. Semantic Web Workshop, 2002.

[28] Kahan, J., Koivunen, M., E. Prud'Hommeaux and R. Swick. Annotea: An Open RDF Infrastructure for Shared Web Annotations. In Proc. WWW 2001.

[29] Kang, H. and B. Shneiderman. Visualization Methods for Personal Photo Collections: Browsing and Searching in the PhotoFinder. IEEE International Conference on Multimedia and Expo (III), 2000.

[30] Miller, R.C. Lightweight Structure in Text. Ph.D. Thesis, Computer Science Department, School of Computer Science, Carnegie Mellon University, May 2002.

[31] North, C. and B. Shneiderman. Snap-Together Visualization: A User Interface for Coodinating Visualizations via Relational Schemata. In Proc. AVI, 2000.

[32] Paine, J. Model Master: An Object-Oriented Spreadsheet Front-end. In Proc. CALECO, 1997.

[33] Quan, D. Designing End User Information Environments Built on Semistructured Data Models. Doctoral Dissertation, 2003.

[34] Quan, D. and D. Karger. How to Make a Semantic Web Browser. In Proc. WWW 2004.
[35] Roth, S.F., P. Lucas, J.A. Senn, C.C. Gomberg, M.B. Burks, P.J. Stroffolino, J.A. Kolojejchick and C. Dunmire. Visage: A User Interface Environment for Exploring Information. In Proc. IEEE InfoViz, 1996.
[36] Sauermann, L. The Gnowsis Semantic Desktop for Information Integration. WM Conference, IOA Workshop, 2005.
[37] schraefel, m.c., Y. Zhu, D. Modjeska, D. Wigdor, and S. Zhao. Hunter Gatherer: Interaction Support for the Creation and Management of Within-Web-Page Collections. In Proc. WWW 2002.
[38] Shih, Lawrence Kai and David R. Karger. Using URLs and Table Layout for Web Classification Tasks. In Proc. WWW 2004.
[39] Shneiderman, B. Dynamic Queries for Visual Information Seeking. Software IEEE, Volume 11, Issue 6, Nov. 1994, pp. 70—77.
[40] Sinha, V. and D. Karger. Magnet: Supporting Navigation in Semistructured Data Environments. In Proc. SIGMOD 2005.
[41] Spenke, M., C. Beilken, and T. Berlage. FOCUS: The Interactive Table for Product Comparison and Selection. In Proc. UIST 1996.
[42] Stuckenschmidt, H., et. al. Exploring Large Document Repositories with RDF Technology: The DOPE Project. Intelligent Systems, IEEE, Volume 19, Issue 3, pp. 34—40, 2004.
[43] Yee, P., K. Swearingen, K. Li, and M. Hearst. Faceted Metadata for Image Search and Browsing. In Proc. CHI 2003.