

User Interfaces Supporting Casual Data-Centric Interactions on the Web

by

David F. Huynh

S.M. Computer Science and Engineering, Massachusetts Institute of Technology (2003)

B.A.Sc. Computer Engineering, University of Waterloo (2001)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2007

[September 2007]

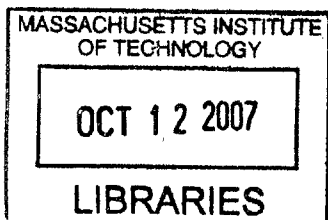
© Massachusetts Institute of Technology 2007. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
August 27, 2007

Certified by
David R. Karger
Engineering Supervisor

Certified by
Miller
Engineering Supervisor

Accepted by
Author C. Smith
Chairman, Department Committee on Graduate Students



BARKER

User Interfaces Supporting Casual Data-Centric Interactions on the Web

by

David F. Huynh

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2007, in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Today's Web is full of structured data, but much of it is transmitted in natural language text or binary images that are not conducive to further machine processing by the time it reaches the user's web browser. Consequently, casual users—those without programming skills—are limited to whatever features that web sites offer. Encountering a few dozens of addresses of public schools listed in a table on one web site and a few dozens of private schools on another web site, a casual user would have to painstakingly copy and paste each and every address into an online map service, copy and paste the schools' names, to get a unified view of where the schools are relative to her home. Any *more* sophisticated operations on data encountered on the Web—such as re-plotting the results of a scientific experiment found online just because the user wants to test a different theory—would be tremendously difficult.

Conversely, to publish structured data to the Web, a casual user settles for static data files or HTML pages that offer none of the features provided by commercial sites such as searching, filtering, maps, timelines, etc., or even as basic a feature as sorting. To offer a rich experience on her site, the casual user must single-handedly build a three-tier web application that normally takes a team of engineers several months.

This thesis explores user interfaces for casual users—those without programming skills—to extract and reuse data from today's Web as well as publish data into the Web in richly browsable and reusable form. By assuming that casual users most often deal with small and simple data sets, declarative syntaxes and direct manipulation techniques can be supported for tasks previously done only with programming in experts' tools.

User studies indicated that tools built with such declarative syntaxes and direct manipulation techniques could be used by casual users. Moreover, the data publishing tool built from this research has been used by actual users on the Web for many purposes, from presenting educational materials in classroom to listing products for very small businesses.

Thesis Supervisors: David R. Karger and Robert C. Miller
Titles: Professors of Computer Science and Engineering

*to my parents
who are my constant source of courage*

Acknowledgements

I would like to thank my thesis advisors, David R. Karger and Robert C. Miller, for guiding me when I was lost and for trusting me to explore freely on my own when I found my direction. I could not have had better supervision for every step of this arduous journey, or more gratification and pride at the end.

The SIMILE project members have also been instrumental. I thank Stefano Mazzocchi, Ryan Lee, Andrew Plotkin, Ben Hyde, Richard Rodgers, V. Alex Brennen, Eric Miller, and MacKenzie Smith for their continuing encouragement and support in various forms as well as their perspectives, insights, and wisdom that bring practicality to my research. Without SIMILE, my research would not have had the same impact and reach.

The Haystack group and User Interface Design group members have provided tremendously insightful feedback on my research, and many of them have been my travel companions throughout this long journey. Many thanks go to Vincet Sinha, Jaime Teevan, Karun Bakshi, Nick Matsakis, Harr Chen, Yuan Shen, Michael Bernstein, Adam Marcus, Sacha Zyto, Max Goldman, Greg Little, and Jones Yu.

I have also enjoyed many exchanges with users of my software published through SIMILE. I thank Johan Sundström, Josh Aresty, and Keith Alexander for their thoughts as well as their code patches, not to mention their enthusiasm in singing to their friends more praises of my work than it deserves. There can be no more satisfaction to a tool builder than to see his tools used for real.

Many people have participated in my user studies. I am grateful for their time and their insightful feedback. I also would like to thank those who have helped recruit subjects for my studies: William Reilly, Robert Wolfe, Ann Whiteside, and Rebecca Lubas.

Over the years, I have enjoyed stimulating conversations with many other people regarding my research. I thank Paolo Ciccarese, Steven Drucker, Mira Dontcheva, Eric Neumann, Daniel Tunkelang, Kingsley Idehen, Ivan Herman, Michael Bergman, Jon Crump, Danny Ayers, Vinay Mohta, Ian Jacobs, Sandro Hawke, Wing Yung, Lee Feigenbaum, Ben Szekely, Emmanuel Pietriga, Chris Bizer, Justin Boyan, Glen McDonald, Michael Bolin, Alex Faaborg, Ora Lassila, Deepali Khushraj, Ralph Swick, Daniel Weitzner, Ann Bassetti, Chao Wang, Gregory Marton, Boris Katz, and Steve Garland.

continued on next page

continued from previous page

My friends have made the journey much more fun. I thank Sayan Mitra, Vineet Sinha, Han-Pang Chiu, Harold Fox, Agnes Ng, Rui Fan, Karun Bakshi, Stefano Mazzocchi, Ryan Lee, Paolo Ciccarese, Heidi Pan, Max van Kleek, Gregory Marton, Michael Bernstein, Olya Veselova, Brooke Cowan, and Jimmy Lin. Special thanks go to 李真 for keeping my company everyday throughout.

Whereas my colleagues asked many deep questions, my family asked a very simple one, “Are you done yet?!” It was deceptively simple: while I could often provide clear responses to my colleagues, I always had to mumble some answer to my family. But now the reply is a resounding “Yes.” I’ve come to the end of my seemingly quixotic journey, and I’m grateful to my family for being so caring and anxious to always ask, “Are you done yet?” but also for being so understanding to never question why.

Whatever you do will be insignificant, but it is very important that you do it.
— Mahatma Gandhi —

CONTENTS

1.	INTRODUCTION	17
1.1	The Case for Data-Centric Interactions	19
1.2	Approach.	21
1.2.1	Assumptions and Limitations	21
1.2.2	Publishing Data	22
1.2.3	Extracting Data	26
1.2.4	Integrating Data	28
1.3	Implications for Semantic Web Research	30
1.4	Contributions	31
1.5	Thesis Outline.	32
2.	RELATED WORK	33
2.1	User Interfaces for Browsing.	33
2.2	Publishing Data.	34
2.2.1	Rationales for Using Data Technologies in Publishing	34
2.2.2	Flexibility of Data Modeling and Presentation	36
2.2.3	Costs of Using Data Technologies	38
2.2.4	Client-side Data Technologies	38
2.2.5	Presentation Templating Technologies	39
2.3	Extracting Data.	40
2.3.1	Goals for Web Data Extraction	40
2.3.2	Supervised vs. Unsupervised Extraction	41
2.3.3	Web Content Augmentation	42
2.3.4	Facilitating Web Data Extraction	42

2.4	Integrating Data	42
2.4.1	Data Integration and Warehousing	43
2.4.2	Ontology Alignment	43
2.4.3	Data Integration User Interfaces	44
2.5	Toward a Data-Centric Browser	45
2.5.1	Data Models	45
2.5.2	Data-Centric Features	45
2.5.3	Deployment Paths	46
3.	PUBLISHING DATA	47
3.1	Interface Design	49
3.1.1	User Interface	49
3.1.2	Publisher Interface	52
3.2	Data Model	55
3.2.1	Items	55
3.2.2	Types	57
3.2.3	Properties	58
3.2.4	Expressions	58
3.2.5	Data Import/Export	61
3.3	User Interface Model	62
3.3.1	Collections	62
3.3.2	Facets	62
3.3.3	Views	63
3.3.4	Lenses	63
3.3.5	UI Contexts and Formatting Rules	64
3.3.6	Coders	67
3.4	Implementation	68
3.5	Evaluation	70
3.5.1	Performance	70
3.5.2	Usage	75
3.6	Summary	76
4.	EXTRACTING DATA	89
4.1	User Interface Design	90
4.1.1	Extraction User Interface	90
4.1.2	Augmentation User Interface	92
4.2	Data Extraction	95
4.2.1	Item Detection	95
4.2.2	Subsequent-Page Detection	97
4.2.3	Field Detection	99

4.3	Evaluation	100
4.3.1	Evaluation of Data Extraction	100
4.3.2	Evaluation of User Interface	100
4.4	Summary	106
5.	INTEGRATING DATA	107
5.1	Scenario	108
5.2	User Interface	109
5.2.1	Creating columns and facets	112
5.2.2	Merging fields	113
5.2.3	Simultaneous editing	114
5.2.4	Faceted Browsing	115
5.2.5	Visualizations	116
5.2.6	Miscellany	116
5.3	Implementation	116
5.4	Evaluation	118
5.4.1	Design and Procedure	118
5.4.2	Participants and Apparatus	119
5.4.3	Results	119
5.5	Summary	121
6.	CONCLUSION	123
6.1	Future Work	124
6.1.1	Publishing Data	124
6.1.2	Extracting Data	124
6.1.3	Integrating Data	125
6.1.4	Toward a Data-Centric Browser	125
6.2	Discussion	126
6.2.1	Data Publishing as Empowerment	126
6.2.2	The Data Market	128
6.2.3	Data as Common Goods	128
6.2.4	Data as Culture	129
7.	BIBLIOGRAPHY	131

1. INTRODUCTION

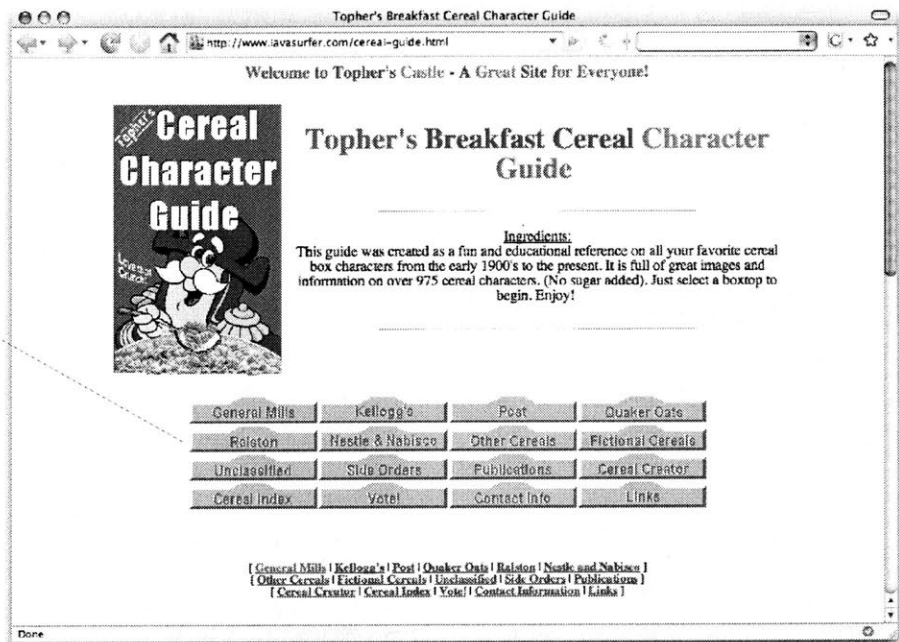
When we encounter data on the Web, most of the time it can only be read on the original page, at the original site, and in its original form. For example, if a few dozen names and addresses are found in a list on a web page, then it is very difficult to plot these addresses all on a map (to visually compare their relative distances to some location) or even just to store their contact information into one's address book. The publisher of that site has decided to present the names and addresses in a list and there is nothing the site's users can do to use that data differently without incurring a great deal of time and effort.

Similarly, it is difficult to publish data to the Web beyond making the data accessible as a spreadsheet or a text file, or encoding the data in hand-written HTML that offers no advanced browsing features or rich visualizations. Consider the home-made site in Figure 1.1 that show breakfast cereal characters. The publisher of that site, Topher, has organized the characters by brand. If Topher decided to let his users browse the characters by year of introduction to market—a reasonable choice, then he would have to completely reorganize his web site. In contrast, the commercial site in the same figure offers several ways of browsing for cameras, and in order to offer more ways, the site can just formulate more queries to its existing database. To catch up with commercial sites, Topher would have to acquire many skills: setting up a database, designing a data schema, designing the user interface, implementing the user interface, implementing the application logic, and testing the resulting three-tier web application on many browsers.

My thesis demonstrates that tools can be built to let casual users—those without programming skills—interact with today's Web in more data-centric ways, to effortlessly retrieve and reuse data from the Web as well as publish data into the Web in a browsable and reusable form. These tools can be built and used today without needing the Web to become more data-centric first by itself.

I. INTRODUCTION

A home-made web site offers only one way to browse the data, and it would be extremely hard for the publisher to offer other ways, as that requires complete reorganization of the web site.



A commercial web site offers many ways to browse the data, and more ways can be added just by formulating new queries to the database.



Figure 1.1. Using just HTML, a small publisher cannot match the advanced browsing features and rich visualizations offered by commercial and institutional sites, or compete with the flexibility and speed at which such features can be implemented by those sites.

1.1 The Case for Data-Centric Interactions

Being able to reuse data on the Web creates new opportunities:

- The same data can show different insights when it is presented differently. Plotting the US presidents on a time line color-coded by their political parties shows the interleaving pattern of Democrats and Republicans coming into power, whereas plotting the presidents on a map by their birth places reveals that all except one were born in eastern United States. Listing the presidents in a table sorted by their surnames highlights two Adams and two Bushes. If a web site provides only one of these presentations and no means to retrieve and reuse its data so that it can be visualized differently, its audience has been denied some of these insights.

- Several sources of data can provide more insights when combined than they can individually. Plotting wars, economic performance, artistic movements, scientific discoveries, etc., alongside the presidents' terms on a time line can reveal each president's and each political party's effects on the country. Plotting cholera cases and water pumps on the same map did bring insight to John Snow, suggesting obvious actions that would stop the cholera epidemic in London in 1854 [72]. As long as it remains hard to combine data from several sources, such insights can easily be missed.

- One important source of data is a user's own information. Seeing several potential houses to buy on a map is useful, but seeing on that same map where one's friends and relatives live and where one's workplace is located can be even more useful in making a house purchasing decision. As often a user's own information is private and cannot be made known to any web site, the data on web sites (which the user is already allowed to see) must leave those sites and come to the user's computer where it can be combined with the user's own information.

These opportunities make the case for getting data on the Web into a form conducive to reuse for casual users' benefits. Once *reusing* data is just as easy as *using* data, there will not be a need to distinguish the two cases. In that future, casual users will always have at hand whatever functionality is appropriate to apply on whatever data encountered wherever on the Web. Such an experience with the Web can be said to be more *data-centric* than *text-centric*.

Not only should casual users be able to easily reuse data from the Web, they should also be able to easily publish data into the Web in ways that satisfy their publishing requirements. For example, if a casual user with no programming experience has gathered all the data about her genealogy and wishes to share that data for her living relatives' enjoyment, she should be able to publish a web site presenting that data. If her family is dispersed geographically, a map would be an appropriate visualization. A timeline can superimpose the lives of her ancestors against a historical backdrop with certain events that she deems important. A thumbnail view shows everyone's portrait together and highlights facial feature resemblance. Grouping or filtering by artistic abilities might reveal which side of

1. INTRODUCTION

her family is more artistic than the other. If many of her kindred have had military careers, browsing and sorting by their military ranks and plotting the military bases where they have served might appeal to her. How the data is displayed and browsed should be left entirely to her—the data owner’s—discretion. Just as she can structure and lay out a *text document* however she wishes using HTML, she should also be able to present her *data* however she likes.

But that is easier said than done. Even to support as basic a feature as sorting a few hundred people by name immediately requires database setup, schema design, and server-side programming—skills that a casual user does not possess and would take years to acquire. Whereas it *was* easy to become a first-class citizen of the *early* Web by authoring HTML in a text editor, it is no longer easy on today’s Web to possess the same publishing power as large companies who have teams of engineers dedicated to building three-tier web applications.

That it is difficult to publish data into the Web and reuse data from the Web is not surprising. The Web was designed for publishing hypertext—*text*—rather than *data*. Web standards (HTML) were specified for publishing text, and web browsers were built for viewing text. Thus we find the majority of web content to be text documents written in natural human languages, unfavorable to data-centric interactions.

This text-centric Web is showing its limits as the demand for data-centric interactions rises. In response, the Semantic Web project [31, 42] holds out the vision of a future Web wherein most if not all data encountered on the Web is described in a standard data model and data-centric interactions can be easily supported. Getting the Web to such a state is a grand challenge, and the slow adoption of semantic web technologies keeps that vision elusive even after a decade of the effort’s existence.

In this thesis, I demonstrate that existing tools designed for experts to publish data to the Web and extract and reuse data from the Web can be adapted for casual users by taking into consideration the casual users’ needs and abilities. These tools allow them to interact with today’s Web in data-centric ways without having to wait for the coming of the Semantic Web.

1.2 Approach

My research approach to providing data-centric interactions on today’s Web to casual users—those without programming skills—can be divided into three aspects:

- publishing data into the text-centric Web and providing data-centric features on that data;
- extracting data from the text-centric Web so that missing data-centric features can be applied; and
- integrating data from several sources to gain value that no single source alone can offer.

1.2.1 Assumptions and Limitations

To scope the research in these three aspects, I have made two assumptions:

- *Casual users most often deal with small data sets, each containing at most about a thousand records.* That is, when a casual user publishes data to the Web, her data most probably consists of no more than a thousand records. When she extracts data from existing web sites, she only extracts from similarly limited data sets. And when she combines data from several sources, she only combines a few sources and each source contains a small data set to begin with. This size limit does not tremendously reduce the benefits of data-centric interactions. For instance, finding the winner in a race among as few as 20 runners is most efficiently and effortlessly done by sorting them by their timings rather than visually scanning over a list of unordered timings. We can also look to the fact that commercial spreadsheet programs have kept their limit of 65,536 rows for over two decades as evidence that up to some size, scale is not relevant to casual users.

- *Casual users most often deal with simple data made up of records consisting of property/value pairs and binary relationships rather than complex class hierarchies with n -ary relationships.* The popularity of commercial spreadsheet programs is evidence that simple tabular data models can carry the usefulness of data-centric interactions a long way for casual users. For instance, being able to align the “home address” field in one table of records with the “residential address” field in another table of records is already useful, such as for plotting all records together on a single map.

In making these assumptions, I have clearly imposed some limitations on my research results. My research contributions may not immediately apply on large, complex data sets, which I argue are rarely encountered by casual users.

1.2.2 Publishing Data

A typical web application consists of three layers: data, presentation, and application logic that connects them. This separation of data from presentation has two advantages.

- First, it allows mixing and matching data with presentation. The same presentation template can be applied to many different data records, making the process of presenting data very efficient. The same data can be shown in different presentations—say, as a list in one case and on a map in another, increasing the richness of the user interface.

- Second, the separation of data from presentation makes it easier to build tools specialized for dealing with either data or presentation rather than both at once. Databases and query languages are designed for manipulating data en-masse while WYSIWYG editors and templating languages are tailored toward specifying visual layouts and styles. It would be much harder to design a piece of software and a standard for manipulating both data and presentation together.

This principle of separating data from presentation has been built into tools and frameworks designed for large publishers—online retailers and institutions—who were the first to run into the need for publishing large data sets and offering advanced features. Built to meet the needs of these large publishers, such as to accommodate complex site structures, to allow for heavily customized looks and feels, and to handle secured online transactions, these technologies are far too complex for casual users to adopt. However, the benefits of separating data from presentation embodied in these technologies are applicable to casual users. Unfortunately, such users have so far only been offered HTML as the generic web publishing technology, and in HTML, data and presentation are mixed together.

To let casual users benefit from the separation of data from presentation, the costs of authoring data, authoring presentation, and connecting them up must be lowered.

The cost of authoring presentation can be lowered in two ways by assuming that the presentation needs of casual users are not so complex:

- First, a variety of common features such as sorting, grouping, searching, filtering, map visualization, timeline visualization, etc. can be provided out-of-the-box so that the user does not need to re-implement them herself. (As her presentation needs get more sophisticated, she can plug in more and more third parties' extensions.)

- Second, customization to the presentation can be specified in an HTML-based syntax right inside the HTML code used to layout the web page. This is so that the publisher can work on every part of the page's presentation inside a single file in a single syntax. (As the publisher's needs to customize the presentation get more sophisticated, the HTML-based syntax will no longer satisfy her.)

The cost of authoring data can be lowered in two ways by assuming that casual users publish only small data sets:

- First, if a publisher is already editing her data in some particular format and some particular editor convenient to her, she should not need to load that data into a database and then maintain the data through the unfamiliar and inconvenient user interface of the database. She should be able to keep managing her data however she likes, and the data only gets loaded into the database when it needs to be rendered. This is realizable if the data is small and loading it into the database is quick. (As the data gets larger, there is a point when the data should already be loaded into a database to ensure a responsive web site.)

- Second, data schemas can be made optional if their benefits do not justify their costs. While schemas are advantageous on large data sets for database optimizations and for managing data at a higher level of abstraction, their benefits on small data sets are much less apparent. (As the data gets larger or more complex, there is a point when schema abstractions benefit both the publisher as well as the publishing engine.)

Finally, all the costs of setting up software (database, web server, and application server) can be eliminated if the software is packaged as a Web API to be included into a web page on-the-fly. Such a Web API can also easily allow for extensions, which accommodate the increasing sophistication in a casual user's publishing needs.

I have built a lightweight publishing framework called Exhibit based on these ideas. To publish an *exhibit*—a web page powered by Exhibit—a publisher uses any text editor to lay out her web page in an HTML file (Figure 1.2) and to enter her data into one or more JSON files (Figure 1.3). The end result is a richly interactive web page such as the one shown in Figure 1.4. If the publisher already has data in some other format, then she can continue to maintain it in that format. Currently, Exhibit can import Bibtex, Excel files, live Google Spreadsheets feeds, RDF/XML, and N3.

When a casual user publishes to the Web using Exhibit, she herself benefits from the separation of data from presentation. Moreover, anyone who visits her web site can easily extract her data for reuse because her data is already in structured form and as publicly accessible as her HTML code. Exhibit kills two birds with one stone: addressing the data publishing need as well as making available in reusable form data that would otherwise be encoded in text-centric HTML.

I. INTRODUCTION



Figure 1.2. An HTML-based syntax is used for configuring the user interface of an Exhibit-powered web page.

```

{
  "items": [
    { "label":      "George Washington",
      "type":      "President",
      "birth":     "1732-02-22",
      "birthPlace": "Westmoreland Country, Virginia, USA",
      ...
    },
    // ... more presidents
  ],
  "properties": {
    "birth": { "valueType": "date" },
    // ... more properties
  },
  "types": {
    "President": { "pluralLabel": "Presidents" },
    // ... more types
  }
}

```

Figure 1.3. Exhibit provides by default a simple JSON syntax for specifying data, but it has an extensible importer architecture for importing other formats including Excel, Bibtex, RDF/XML, N3, and live Google Spreadsheet feeds.

The screenshot displays a web browser window with the title "SIMILE | Exhibit | Presidents (select Birth Places view)". The address bar shows the URL "http://127.0.0.1:8888/exhibit/site/examples/presidents/presidents.html". The main content area is titled "US Presidents" and includes a sub-header "Here is the Exhibit JSON data file." Below this, there are navigation links: "BIRTH PLACES", "DEATH PLACES", "TERMS", "TABLE", "DETAILS", and "PHOTOS". A map of the United States is shown with circular markers representing the birthplaces of 42 presidents. The markers are clustered in the Northeast and Midwest. To the right of the map, there are three filter sections: "Search:", "Religions" (listing Anglican, Baptist, Church of Christ, Congregationalist, Deism, and Deist), "Political Parties" (listing Democratic, Democratic-Republican, Federalist, No Party, Republican, and Whig), and "Died In Office" (listing no and yes). The browser status bar at the bottom indicates "Transferring data from maps.google.com...".

Figure 1.4. A web page powered by Exhibit provides text search and filtering features as well as rich visualizations such as maps and timelines.

1.2.3 Extracting Data

Just like conventional publishing technologies, conventional web data extraction technologies are unsuitable for casual users as they also have been designed for large data sets and complex use cases. If a casual user encounters a few dozen street addresses on a site that offers no map feature, she will not spend much effort to learn these advanced technologies to scrape the data and then build a “Web 2.0 mash-up” site just to map those few dozen addresses.

Web data extraction has been mostly employed by web sites that scrape other web sites for data, aggregate the data, and then offer new services on that data. It makes sense that the scraping be tailored to each source web site to ensure the quality of the result, and that the scraping then be automated to keep the data up-to-date in an efficient manner. As the data is offered again in completely new web sites, there is no need to retain the presentation elements of the original sites.

Although casual users also make use of web data extraction technologies for repurposing data on the Web, their requirements differ from those of large data aggregation sites. The typical task of, say, an online flight reservation site is to accumulate as complete and up-to-date as possible flight information from several airlines, amounting to thousands of records. In contrast, a casual user might just want to pull out street addresses of a few dozen of private schools from their school board web site, plot them on a map, make a one-time printout, and not bother to update the map ever again. That is, the user deals with a lot less data, cares only for a few fields (e.g., street address) rather than all fields, and does not need to keep the extracted data up-to-date since she only needs the printout once. Thus, casual users might put less demand on web data extraction algorithms with respect to scalability, accuracy, and automation.

In other aspects, however, casual users might have more demands than large sites. Skills and resources that a data aggregation site has at hand to repurpose scraped data into a new three-tier web application are not available to casual users: no casual user can be expected to set up, design, and maintain a database, to design and implement a rich user interface, and to connect them with application logic, *especially just to plot a few dozen street addresses on a map*. Thus, web data extraction tools built for casual users must offer as complete an experience as possible. Instead of simply returning raw data, they must behave like web applications themselves, offering appropriate presentations and features that casual users need to accomplish their tasks. That is, it is more about adding in missing features rather than taking out data.

To offer a complete web application-like experience over extracted data, all three layers of a typical web application must be automated as much as possible.

- First, the user interface can be “designed” with zero user intervention just by reusing as much as possible the presentation elements already in the original web site. In particular, when a data record is extracted from a part of an original web page, that fragment of the web page is also extracted so that in order to show that data record later on, we can simply show the fragment again. The rest of the web

page, which does not contain data to extract, can also be kept as-is so to preserve the original visual context. This is novel since existing web data extraction algorithms throw away original presentation elements (because they are not needed for the purposes of large data aggregation sites).

- Second, in the application logic layer, some set of commonly needed features such as sorting and filtering can be provided out-of-the-box so that there is no need for a casual user to implement them.

- Finally, the extracted data can be loaded immediately into a database without any user intervention if the extracted data can be used as-is without further processing by the other layers. The biggest roadblock to using the extracted data as-is seems to be the need to label fields because field labels are used in conventional user interfaces to provide affordance for features like sorting and filtering (e.g., “sort by publication date”). Unfortunately, field labels can be hard to recover from

The screenshot shows the Amazon.com search results for "jeffrey archer" in Mozilla Firefox. The page displays search results for books, including "False Impression", "The Eleventh Commandment", and "Kane & Abel". Two Sifter overlays are visible on the right side of the page, providing sorting and filtering options for the search results.

Explore (Items filtered from 48 original items)

27 (Previous 16 Next 16 >>)

Pages: [1] 2 >>>

Click to sort items: ascending descending original order

Select to filter items: unselect all

Value	Items
Audio Cassette	1
DVD	0
Hardcover	8
Paperback	19

Click to sort items: ascending descending original order

Select to filter items: unselect all

Value	Items
2000 to 2010	11
2001	1
2003	3
2004	2
2005	4
March 2005	1
July 2005	2
November 2006	1

An asterisk is inserted after each field value. Clicking on an asterisk adds sorting and filtering controls for that field.

Figure 1.5. By keeping presentation elements from the original web site, Sifter can apply direct manipulation techniques on the extracted data fields to support sorting and filtering without requiring the user to label the fields.

web pages. For example, in Figure 1.5, nowhere on the web page says that “Mar 7, 2006” is the “publication date” of “False Impression.” However, direct manipulation techniques can be applied to avoid the need for field labels altogether. If the user can interact directly with the text “Mar 7, 2006” to invoke sorting and filtering operations, it does not matter if “Mar 7, 2006” is the publication date of the book or the birth date of the author. It is *just* a date field and it can be sorted and filtered mechanically without any regard for its actual semantics.

I have built these ideas into a browser extension called Sifter (Figure 1.5) that can augment a web site *in-place* with filtering and sorting functionality while requiring from the user as few as two clicks. The added features work inside the site’s own pages, preserving the site’s presentational style, as if the site itself has implemented the features.

1.2.4 Integrating Data

Data integration tools have also been built for a different audience than casual users. For a typical conventional data integration task, such as merging huge databases of two institutional libraries together to serve the combined data through a new web site, several experts and programmers are involved to handle different aspects of the task, including aligning the schemas, cleaning up the data, and then building the web site. Each tool employed is specialized for one stage of the process and designed to handle large, complex data sets. In contrast, a casual user might just want to merge two lists of a few dozens of addresses from two web sites together to plot them on a common map. There is much less data and the data is much simpler. Power tools used by experts are too advanced for casual users.

Whereas a team of experts and programmers can work most efficiently by dividing a data integration task into stages and letting each team member specialize on one stage, each casual user only has herself to deal with her own data integration task. She needs not a set of highly specialized tools like those for experts but a single tool that lets her work on the whole task. Furthermore, unlike experts experienced in dividing a clearly defined task cleanly into stages, the casual user might have to deal with different aspects of the task in an interleaving manner, switching from cleaning up data to constructing presentation and back again, as she gains more and more understanding of what she needs, of what the data is like, and of how to handle the task.

Compared to experts, casual users lack both data modeling skills and programming skills. However, for small, simple data sets, neither skill set may be crucial. First, when the data set is small, schemas—useful for efficiently reasoning about a massive quantity of data in the abstract—are not as useful and can introduce overhead cognitive costs. Instead of having to learn about theoretical concepts like schemas, casual users can manipulate data instances directly. Second, instead of using programming to process data, casual users can just use direct manipulation

techniques. For example, two fields can be aligned by dragging and dropping one onto the other.

I have demonstrated these ideas in a tool called Potluck that lets casual users pool together data from several sources, supports drag and drop for merging fields, integrates and extends the faceted browsing paradigm for focusing on subsets of data to align (Figure 1.6), and applies simultaneous editing [59] for cleaning up data syntactically (Figure 1.7). Potluck also lets the user construct rich visualizations of data in-place as the user aligns and cleans up the data. This iterative process of integrating the data while constructing useful visualizations is desirable when the user is unfamiliar with the data at the beginning—a common case—and wishes to get immediate value out of the data without having to spend the overhead of completely and perfectly integrating the data first.

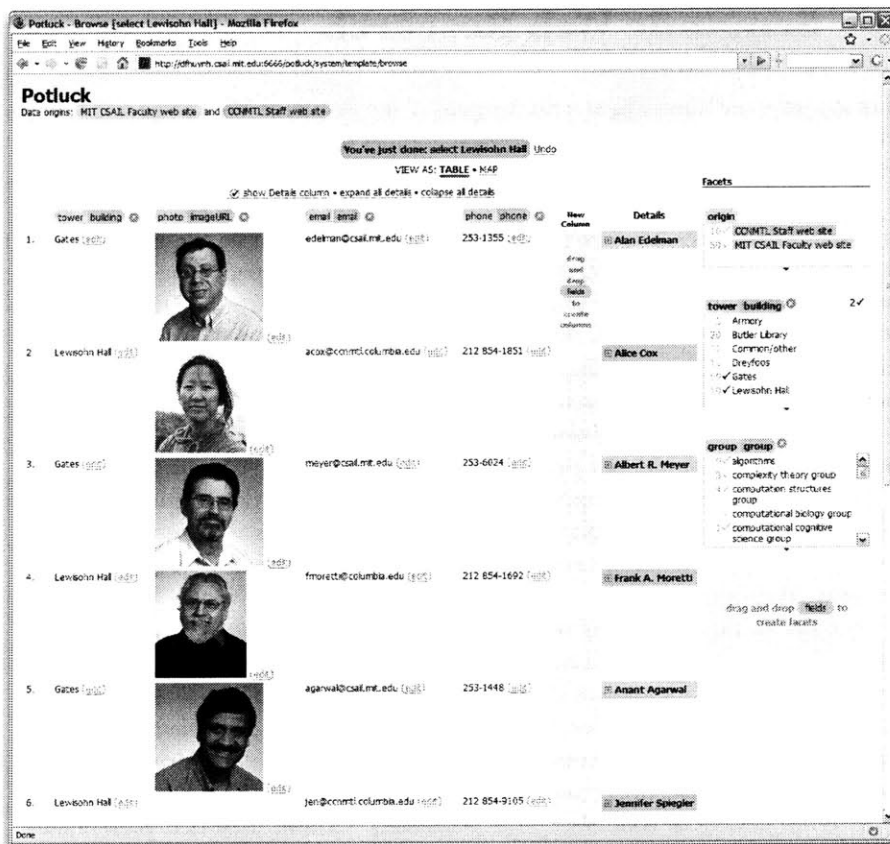


Figure 1.6. Potluck supports drag and drop for merging fields and constructing visualizations. Faceted browsing is available at the same time to help the user isolate subsets of interest or subsets that need further cleaning or alignment.

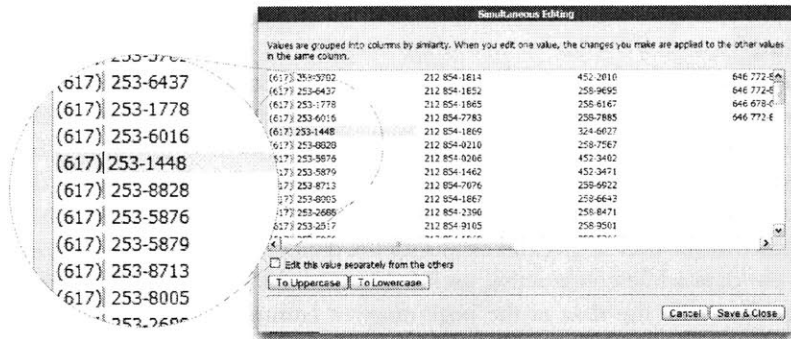


Figure 1.7. Potluck applies the simultaneous editing technique to let the user clean up data syntactically in an efficient manner.

1.3 Implications for Semantic Web Research

In many ways my thesis aligns with the goal of the Semantic Web project [31, 42]: both aim for a data-centric Web. The philosophies in my research approaches can bring insights to the Semantic Web research community:

- *Before making data meaningful to machines, make it meaningful to people.* In its early days, the Semantic Web research community focused most of its efforts on issues in data modeling, artificial intelligence and agent automation, neglecting to build user interfaces for humans to use semantic web data. Even today, the latest semantic web browsers show raw URIs that make no sense to casual users. In contrast, my research starts out by examining the needs and characteristics of casual users so to build data-centric interfaces that make sense to them.

- *In addition to building a data-centric Web that benefits humanity decades into the future, build data-centric technologies that benefit some individuals right now.* Addressing real needs of people right now motivates the creation of that future data-centric Web and helps identify real needs in that future. If nothing else, this strategy allocates resources to research on data-centric user interfaces, which are severely lacking in Semantic Web research.

- *Before making data reusable, make it useful.* Whereas the Semantic Web researchers encourage people to publish data for the sake of future reuse by other people, my Exhibit framework encourages people to publish data just because publishing data separate from presentation makes the publishing process efficient and the results richly interactive. Immediate, concrete, personal usefulness drives adoption more than prospective, uncertain benefits to someone else.

- *While focusing on data, don't forget presentation.* Exhibit uses rich presentation (map, timeline, etc.) as the motivation for publishing data. Sifter keeps existing presentation elements from original web pages to preserve visual context as well as support direct manipulation on the scraped data, saving casual users from labeling

fields. Potluck lets casual users use the presentation they are constructing to guide their data integration process: if the aligned data “looks right” in the user interface, it should be right in the data model.

- *Even while aiming for semantics, take care of syntax.* Exhibit’s HTML-based presentation configuration syntax and its ability to import many data formats lower the barrier to its adoption. Sifter’s ability to recognize dates and numbers makes its sorting and filtering features more useful. Potluck’s simultaneous editing feature lets casual users efficiently fix up data syntactically.

1.4 Contributions

My thesis statement is:

Data-centric interactions with today’s Web are useful to and feasible for casual users, and usable tools can be built to support such interactions by gearing for small, simple data sets and common casual needs.

This thesis makes the following contributions:

- First, this thesis combines a simple graph-based data model with simple extensions of the HTML syntax in order to let casual users—those without programming skills—publish web pages offering advanced browsing features and rich visualizations.
- Second, this thesis shows that the cost of using web data extraction technologies can be lowered for casual users by retaining presentation elements from the original web pages. These elements preserve visual context and visual semantics so that direct manipulation techniques can be applied to augment the original pages with more features without requiring users to label fields in the extracted data.
- Third, this thesis proposes that for simple data sets, direct manipulation techniques can be used to let casual users integrate data and construct rich visualizations from it without any need for programming.
- Finally, by letting casual users publish data to the Web in browsable form, extract and reuse data from the Web, and integrate data from several sources without any need for programming, this thesis demonstrates that data-centric interactions can be offered to casual users on today’s Web without having to wait for a semantic web.

1.5 Thesis Outline

Following this introduction, chapter 2 surveys related work. Then, the main body of this dissertation delves into the three areas identified in the approach:

- Chapter 3 details the experiences of using Exhibit to publish as well as using Exhibit's features on Exhibit-based sites. The chapter also outlines Exhibit's architecture and analyzes its actual use.

- Chapter 4 describes how Sifter delivers web data extraction technologies into the hands of users with no programming experience. Sifter was tested on real web sites and real users, and the results indicated that people could use Sifter to augment existing web sites and then perform sophisticated queries and high-level analyses on sizable data collections.

- Chapter 5 motivates Potluck with a data integration scenario and then explains how Potluck's user interface design meets the needs in that scenario. The chapter also details Potluck's implementation as well as reports the results of a user study that identified usability successes and problems.

Finally, chapter 6 reviews the contributions of this dissertation and outlines future directions for this research.

2. RELATED WORK

In every part of my research, browsing is a feature advocated to be useful to casual users. Browsing is thus factored out and discussed first as a topic by itself. Following this topic, three sections discuss work related to the three core components of the thesis: publishing data, extracting data, and integrating data. Finally, this chapter ends with a section on previous approaches to building a data-centric browser.

2.1 User Interfaces for Browsing

To allow users to browse through a data corpus, traditionally the data is organized into a hierarchy of some forms and then links are shown for drilling down the hierarchy. Such hierarchies were called “directories” on portal sites such as Yahoo! in the early days of the Web.

Faceted browsing was pioneered by R.B. Allen [39] in 1995 for browsing document collections that fall into several orthogonal sets of categories—or *facets*—which do not naturally fit together to form a single hierarchy. For example, data on schools can exhibit several facets: locations of the schools, subjects in which the schools excel, sport team performance, special facilities, etc. Although most parents looking for schools for their children probably start by filtering the schools by location, a parent with a child gifted in Math probably wants to filter the schools by subject instead; and a parent with a physically disabled child probably thinks first about special facilities. Picking any one hierarchy will make it unnatural and inefficient for some of the users to browse the data. In the faceted browsing paradigm, all sets of categories—all facets—are offered simultaneously so that each user can start filtering on any facet and continue to filter by any other facet subsequently.

2. RELATED WORK

In effect, faceted browsing lets each user build her own hierarchy on the fly as she picks which facet to filter by at each step of her browsing process. Faceted browsing has been adopted in many online shopping sites because different users have different criteria for choosing products.

Faceted browsing was subsequently adopted for browsing data collections on web sites by J. English, M. Hearst, R. Sinha, K. Swearinen, and K.P. Yee in an unpublished manuscript in 2002 [47].

Faceted browsing is even more useful in tools intended for browsing arbitrary structured data, as there is no way to create a organization hierarchy for that data beforehand. This is why many generic semantic web browsers adopt faceted browsing [53, 60, 64, 68].

All faceted browsing interfaces that I am aware of display preview counts next to each choice in each facet. This is not novel in faceted browsing but a contribution brought over from query preview interfaces [61]. These counts give users information about the result set of each choice before the choice is clicked, saving users from going down any obviously wrong path and having to back up. These counts in each facet also serve to summarize the data by showing the distribution of the data values in that facet.

My work extends the faceted browsing paradigm in a few ways. Sifter offers faceted browsing without field labels and shows that it is workable. Potluck tailors for data cleaning and integrating tasks by adding to each facet a way to filter down to records that are missing data for that facet. Exhibit lets casual users create faceted browsing interfaces by themselves.

2.2 Publishing Data

This section surveys the technologies for and research on data publishing from several perspectives. First, the purposes of using data technologies in the publishing process are discussed: data can make the publishing process more efficient and the outcomes better, or data itself is the outcome that, for instance, serves policies of open data access and visions of data interoperability. Understanding what a data publishing technology is designed to do helps assess its effectiveness. Second, data publishing technologies and research tools are analyzed by three criteria: the flexibility they afford publishers to model data, the flexibility they allow publishers to specify presentations, and the efforts required of publishers. The section then concludes with the discussion of previous work related to specific components of Exhibit, specifically its client-side database, its expression language, and its lens templating facility.

2.2.1 Rationales for Using Data Technologies in Publishing

People are either motivated to use data technologies in publishing because those technologies make the publishing process itself better in some ways, such as more efficient, or because those technologies produce data useful for some other purposes.

2.2.1.1 *Direct Benefits*

By the principle of separation of concerns, database technologies, complemented with server-side templating technologies such as ASP, PHP, and JSP, let online retailers and institutional publishers publish large quantities of information efficiently and offer data-centric features on their web sites. Database technologies are crucial for implementing data-centric features such as sorting, searching, and filtering. Templating technologies let the presentation of a whole web site be made coherent easily and quickly. Recently, client-side templating technologies—XML together with XSLT, JSON [13] together with mjt [18] and the like—are also used toward the same goal. Data technologies are used in this common case as a means to make the publishing process efficient and the outcomes visibly better. The motivation is economic and well understood. Exhibit serves the same motivation but targets casual users instead of large publishers, leading to different design criteria, such as trading off scalability for ease of use.

2.2.1.2 *Indirect Benefits*

Another goal for using data technologies is to enable prospective uses of and benefits from one's own data. In certain cases the prospective use is well understood: RSS enables news articles and blog posts to be aggregated by topics to meet the tastes of individual users. For another example, Mangrove [58] is a project that aims to aggregate RDF annotations on web pages published by individuals within a department to automatically construct departmental directories and calendars. (RDFa [27] and eRDF [25] are also syntaxes for embedding RDF within web pages.) Recently, microformats [17] are recommended for marking up semantic tidbits within web pages so that microformat-aware browsers can provide additional features on those tidbits, such as a contextual menu command for dialing phone numbers.

In other cases, prospective uses are more experimental: recently, many web sites expose web APIs for users and third parties to explore potential mash-ups for their data and services. Microformats could also be said to support experimental prospective uses because until they are widely adopted, their actual use is still to be determined. RDFa and eRDF, advocated to serve the same purpose as microformats, are similarly experimental. In fact, being more general than microformats and more cumbersome to write, the usefulness of RDFa and eRDF over microformats is still a debate.

In the rest of the cases, prospective uses for published data are highly speculative: the Semantic Web project [31] and related efforts, such as the Linked Data

best practice [14], advocate publishing of interlinked semantic web data to enable unexpected reuse, explaining that “it is the unexpected reuse of information which is the value added by the web” [14]. While I do believe in the value of information reuse, I doubt that many individuals can be easily persuaded to labor altruistically for unpredictable future benefits to humanity. Thus, prospect for information reuse cannot itself be an advocacy but must be a side effect of some other concrete and near-term benefits. Failing that, the publishing of data must be enforced by policies of some governing bodies, as is the case with the PubMed service offered by the United States National Library of Medicine [24]. Still, publishing data for no particular use is challenging as there can be no practical criterion by which the quality of the data and of the publishing process can be judged.

2.2.2 Flexibility of Data Modeling and Presentation

Data publishing technologies can be compared along two orthogonal dimensions: flexibility of data modeling and flexibility of presentation. The former is a spectrum ranging from rigid schemas that cannot be changed to general data models that can fit any information. The latter ranges from predetermined layouts and styles to completely customizable presentations. Figure 2.1 lays out this space and frames the discussion that follows.

2.2.2.1 Domain Specific Approaches

Rigid schemas are often enforced by domain-specific applications and web services. For example, web photo album generating tools and hosting services such as Gallery [9] and Flickr do not allow any field beyond the typical EXIF headers, date/time, title, description, tags, and geographical coordinates. If a chef wants a schema to record the main ingredients and cuisines in the photos of her dishes, she cannot extend Gallery’s and Flickr’s schemas properly but would have to coerce the generic tagging mechanism for that purpose. However, the two fields “main ingredient” and “cuisine” will be mixed together, with “Pork” and “Portuguese” next to each other if the tags are sorted alphabetically by default.

Their lack of data-modeling flexibility aside, domain-specific tools do often offer well-designed presentations suitable for their domains. Built on fixed schemas, it can also be easy for them to provide a theming framework for third parties to customize their look and feel. Theming might involve anywhere from overriding CSS styles to editing HTML layouts to server-side programming.

2.2.2.2 Domain Generic Approaches

Domain-generic applications and services are built to support arbitrary schemas from the ground up. Online spreadsheet services, such as DabbleDB [5], EditGrid [21], and Google Spreadsheets, work just like desktop spreadsheet applications and can hold any sort of data. As a one-size-fits-all online database service, Google Base [10] holds topics ranging from local events to weight loss programs. Similarly, Freebase [8] strives to be “a global knowledge base: a structured, searchable,

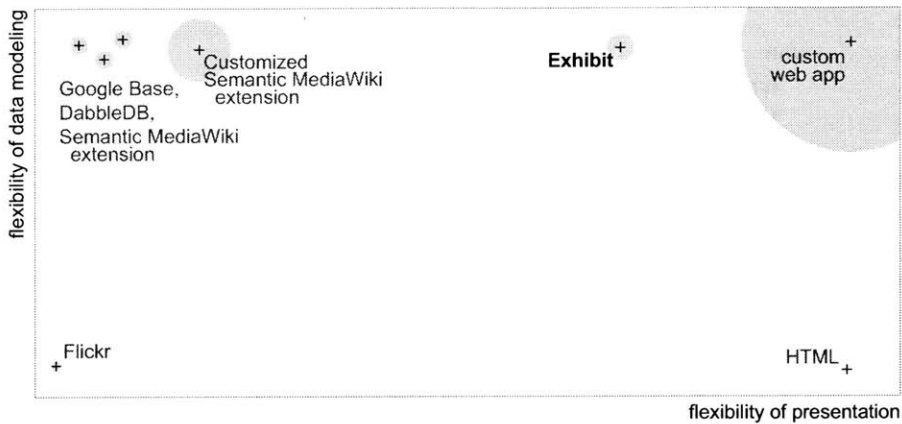


Figure 2.1. Flexibility of presentation and data modeling as well as the efforts required to adopt and use (circle size) for various publishing frameworks.

writeable and editable database built by a community of contributors, and open to everyone,” which means that it has to hold data on any topic.

Domain-generic data modeling standards, specifically RDF [28] which is designed for the Web, also strive to allow for arbitrary data models. The Semantic MediaWiki extension [73] described previously is built on RDF and can thus power wikis containing data on any topic. Various RDF-based browsers, such as mSpace [64], /facet [53], and Longwell [15], provide advanced browsing and visualization features on arbitrary data, albeit in generic ways. (Hildebrand [30] provides a survey of many existing RDF-based browsers.)

The flexibility in data modeling is often correlated with generic user interfaces: Freebase, Google Base, Semantic MediaWiki extension, online spreadsheet services, and most RDF-based tools present their data in generic property/value tables that are not optimized visually per domain. For example, when displaying the information on a paper, it is sufficient and conventional to show:

Huynh, D., Miller, R., and Karger, D. Exhibit: Lightweight Structured Data Publishing. WWW 2007.

but domain-generic tools would instead display:

```

type: Paper
title: Exhibit: Lightweight Structured Data Publishing
author:
  type: List
  element 1:
    type: Person
    title: Huynh, D.
  element 2:
    type: Person
    title: Miller, R.
  element 3:
    type: Person

```

2. RELATED WORK

title: Karger, D.
venue:
type: Conference
title: WWW
year: 2007

The result is at best a waste of space and at worst a cognitive load on the reader to visually parse unnecessarily spaced out and redundant information. Such presentations are often a concession that frameworks to support specifying custom presentations on domain-generic information are hard to design and implement. If customization is allowed, it often requires programming rather than WYSIWYG editing, such as in the use of Fresnel [44] in the Longwell RDF browser [15]. Fresnel is a vocabulary for displaying RDF, and specifying presentations of RDF using Fresnel involves coding in some RDF syntax such as N3. My Exhibit framework also requires coding and its differences with the other frameworks will be discussed section 2.2.5.

While presentations on domain-generic web services are impossible to customize if the services do not offer a customization mechanism, as is the case with Google Base and Freebase, domain-generic tools such as Semantic MediaWiki extension can be customized at the cost of some server-side programming.

2.2.3 Costs of Using Data Technologies

Data technologies for publishing can also be assessed by how much efforts they require of publishers. Web services like Google Base and Freebase only require account registration and familiarization with their web interfaces. Faceted browsing applications such as mSpace [64], /facet [53], and Longwell [15] require downloading the software, installing it (e.g., setting it up on a web server), configuring it (e.g., setting up its database), and loading data into it possibly through programming or command line interfaces.

2.2.4 Client-side Data Technologies

One of Exhibit's distinctive characteristics is its use of a client-side database to simplify the publishing process, eliminating server-side setup and configuration as well as allowing the data to be authored in any format and in any software, and then imported at the last moment when the web page is loaded into the browser.

Client-side data technologies are not novel. In the simplest case, data can be downloaded to the client side as XML documents and queried using XPath. Tabulator [43] is a web application designed to load more and more RDF data from arbitrary web sites to let the user explore the Semantic Web by following RDF links within data from site to site. Tabulator works by dynamically loading RDF data into a client-side RDF store also implemented in Javascript. TrimQuery [37] is another client-side database that supports a SQL-like query language.

Exhibit’s expression language, designed for retrieving data from Exhibit’s database and optionally performing calculations on that data, resembles many proposed RDF path languages [26], which in turn mimic the XPath language. Compared to these languages, Exhibit’s expression language currently lacks conditionals within paths—the ability to specify a boolean expression on a segment on a graph path to filter the candidate result nodes on that segment to only those for whom the expression is true. Conditionals are useful for such a case as, given a person, querying for her children under 20 years of age. However, Exhibit’s expression language can be extended with new functions to support domain-specific computations, such as geographical distance calculations.

2.2.5 Presentation Templating Technologies

Presentation templating technologies range from straightforward substitution of data values into forms (filling in the blanks) to recursive rule-based view resolution to completely automated schema-based UI generation.

Server-side templating languages such as ASP, PHP, and JSP and their client-side counterparts such as mjt [18] are used for straightforward form-based substitution. To create the presentation for a data record, an HTML page or fragment is written, laying out the structure of that presentation, and wherever data should appear in the HTML, calls to the back-end database are inserted instead of actual data values. Branching and looping constructs are supported for more involved cases, such as for rendering a list of several authors of a paper. Such constructs make these approaches less descriptive and more procedural.

The form-based substitution approach is simple to use but does not facilitate reuse of fine-grained presentation logic. For example, if books and papers are to be presented sufficiently different so to warrant different templates, then whatever about books and papers that should still be shown in the same way must be duplicated in both templates. For instance, if authors should be shown as “last name, first initial” in comma-separated lists for both books and papers, then the code for doing so must exist in both templates. To facilitate reuse, view systems, such as the Haystack’s user interface framework [62] and Fresnel[44]-based interfaces, stitch together fine-grained presentation logic based on presentation rules that match each piece of data to be displayed with the appropriate piece of presentation logic in a given context. In the “book” and “paper” context, lists of authors will be matched with the same presentation logic, resulting in the same presentation. That piece of presentation logic needs to be written only once and then registered for both contexts.

While view systems are more powerful than the form-based substitution approach, especially when applied on complex data models, they are also harder to use. The presentation of a piece of data, embedding the presentation of other pieces of data related to it, is not specified in a single block of code, but dynamically composed by evaluating possibly complex and conflicting rules, pulling together different fragments of presentation logic.

As Exhibit is intended for small data sets, it adopts the form-based substitution approach, trading off power for simplicity.

2.3 Extracting Data

There are many goals for extracting data from the Web. This section will first discuss those goals before diving into the two main families of approaches to web data extraction: supervised and unsupervised. The section will then discuss the literature related to the primary goal of web data extraction for casual users: augmentation of web user experience. The section concludes with a survey of efforts that intend to facilitate web data extraction.

2.3.1 Goals for Web Data Extraction

The most common goal for large-scaled web data extraction is to fuel some systems with the data. Online flight booking services such as Expedia, Travelocity, Kayak, etc., scrape airline sites for flight data and aggregate it to provide one-stop flight shopping experience over many airlines. There are similar aggregation services for commercial merchandise, such as Google Product Search. Research projects such as START [33] scrape the Web for facts to answer natural language questions.

In other cases, what the extracted data will fuel is less clear. Swoogle [34] crawls the Web for semantic web documents, purporting to be “Google for the Semantic Web” but there is no investment in the usability of its user interface, making its purpose unclear. DBpedia [7] scrapes templates in Wikipedia to accumulate an RDF data set of millions of triples of facts, and YAGO [69] scrapes Wikipedia’s text to derive millions of relationships. Freebase [8] also scrapes Wikipedia in order to bootstrap its own database. Without concrete uses for their data, it is unclear how DBpedia, YAGO, and Freebase can objectively assess the data and their extraction processes.

Web data extraction is also used for augmenting web user experience. Faaborg’s and Lieberman’s Miro system makes use of data detectors to extract tidbits out of web pages and then applies automations over them as have been demonstrated by example through the complementary system called Creo [48]. Using Creo and Miro, users can automate tasks that would require repeating the same sequences of actions through web sites, such as ordering items on a grocery shopping list by demonstrating how to order one of them. Marmite [75] specializes in letting casual users build web mash-ups by visually stringing together data processing elements and filling out their parameters.

Often the web user experience augmentation is simply the ability to bookmark or save fragments of web pages. Hunter-Gatherer [65], the Internet Scrapbook [70], and commercial tools like NetSnippets [19] let users clip out, collect, organize, and make reports out of web page fragments. Thresher [54], the browser

extension of Dontcheva, et al. [46], and the AdaptiveBlue browser extension [1] even extract structured data from web pages, such as books' titles, authors, publishing dates, and keywords.

2.3.2 Supervised vs. Unsupervised Extraction

Web data extraction approaches fall on a spectrum from entirely supervised to entirely unsupervised algorithms. There are many unsupervised algorithms, relying on partial tree alignment [78, 66], tree edit distance [63], and tabular structures [57] to isolate data records from web pages. Wang et al. [74] claim to be able to label the extracted fields from web sites that offer complex search forms by watching where the field values programmatically entered into the forms reappear in the search result pages. In general, research efforts that yield unsupervised algorithms are focused mainly on the algorithms themselves and are detached from how their algorithms and resulting data can actually be used by human users.

Supervised algorithms require user intervention and thus need user interfaces. With users' help, they are also more accurate than unsupervised algorithms and are more suitable for producing data that will actually be used immediately by users. Thresher [54] allows a user of the Haystack system [62] to mark out a sample data record within a web page and label the fields within that record. Thresher then learns the patterns for extracting the rest of the records on the page. The browser extension by Dontcheva et al. [46] works in a similar way, but while Thresher allows the user to use arbitrary schema to label fields, Dontcheva's system is limited to only a dozen fields. On the other hand, Dontcheva's system lets an existing extraction pattern be updated iteratively when the user returns to an old page and marks up more fields. Dontcheva's system can also scrape detailed pages linked off from the original page to scrape.

Recently, there are web applications such as Dapper [6] that let users scrape existing web sites for data and serve that data up in structured formats as "feeds," or make use of data already scraped by other people. These web applications still offer very limited capabilities for cleaning up data and constructing rich visualizations. Those that offer more capabilities, such as Ning [20], require programming.

While user supervision makes the extraction algorithm more accurate, it puts more demand on the user and increases the cost/benefit ratio for using the extraction tool. Dontcheva et al. report that users of their system often forgot to label fields.

In contrast to all of these supervised and unsupervised approaches, my tool Sifter shows that field labels are not needed before some value can be added on the extracted data. Sifter retains the presentation elements in the original web page and uses them to visually bind augmentation features to the extracted fields without needing field labels.

2.3.3 Web Content Augmentation

In the early days of the Web, there were several research projects, such as WBI [41] and WebWatcher [55], that augment web pages with navigation recommendations by watching the user's browsing actions. Microsoft's SmartTags were originally designed for adding contextual menu commands to semantic tidbits detected in web pages, letting users right-click on any piece of text that looks like a phone number and choose the "dial" contextual menu command. All of these tools work by injecting additional content into the original web pages or hooking into their contextual menus.

Recently, Thresher [54] scrapes web pages and then offers appropriate contextual menu commands whenever the user right-clicks on a data record on an already scraped page. Thresher also delegates other augmentations to Haystack. For instance, scraped data records can be browsed through Haystack's faceted browsing interface.

There are also programming tools like Greasemonkey [12] and Chickenfoot [45] that enable users to script modifications on web pages, but so far they lack a rich data model to support augmentations more sophisticated than just cosmetic changes (e.g., removing ads) and simple addition of third-party content to web pages (e.g., injecting prices from competing sites).

Sifter is novel in its choice of faceted browsing as a useful augmentation and in its ability to offer faceted browsing features in-place over sequences of web pages.

2.3.4 Facilitating Web Data Extraction

A few efforts have advocated the embedding of semantic markups within HTML code so to facilitate web data extraction. The Mangrove project [58] "seeks to create an environment in which users are motivated to create semantic content because of the existence of useful semantic services that exploit that content and because of the process of generating such content is as close as possible to existing techniques for generating HTML documents." Mangrove provides web services such as a departmental calendar, a departmental personnel directory, a semantic search service, etc., that are all fueled with data extracted from web pages annotated with RDF. RDFa [27], eRDF [25], and microformats [17] are standards intended for the same purpose, but at the scale of the whole Web rather than one department.

2.4 Integrating Data

This section starts by reviewing literature in the database field on data integration and data warehousing, which are two families of approach for pooling data from several sources and providing a coherent access point to all of them. One aspect of

data integration is ontology alignment, which will be discussed next. Finally, related work on data integration user interfaces is examined.

2.4.1 Data Integration and Warehousing

Pooling data together from several sources to create a coherent view can be done in two main families of approach called *data warehousing* and *data integration*. Data warehousing involves the *Extract, Transform, and Load* (ETL) process: extracting data from external sources, transforming it to fit internal needs, and loading it into a new database. Once loaded, applications can then be written on the new database, which can be updated periodically to reflect changes to the original sources. Data integration, also known as data federation, involves dynamically querying the original sources and integrating the search results on-the-fly to fit a *mediated schema*. Both approaches have their own advantages and disadvantages, such as whether the data is kept up-to-date, how real-time queries can be answered, and how costly it is to add more sources [51].

Since the 1990s, data integration approaches have developed into an industry called Enterprise Information Integration (EII). It also has a sister industry called Enterprise Application Integration (EAI) which focuses on getting applications to connect, rather than data sources to connect. The customers of these industries are businesses rather than consumers. These industries are still not yet mature as argued by a panel of experts at as recently as SIGMOD 2005 [50].

There is a related industry called Enterprise Analytics (EA) aimed to provide tools for discovering insights over several sources of data internal to businesses. For example, Spotfire [32] couples advanced browsing features such as dynamic query filters with rich visualizations such as starfield displays to help detect outliers and highlight trends. Compared to EII and EAI, EA is also selling to business but its products are positioned much closer to human users. All of these industries produce tools for experts and make trade-offs that favor the experts' needs. For instance, the ability to handle complex ontologies will be favored over the ease of learning how to use the tools.

2.4.2 Ontology Alignment

One aspect of data integration involves aligning original ontologies with one another, matching up classes and properties, or alternatively mapping each of the original ontologies into a mediated ontology. This topic of ontology alignment is also known as ontology mapping, ontology merging, and ontology integration. Kalfoglou and Schorlemmer conducted a comprehensive survey of ontology mapping research in 2005 [56], reporting on a total of 35 works. These works are either tools for data modeling experts or domain experts, or machine learning algorithms, or combinations of tools and automation. They target large and intricate ontologies for which data modeling expertise is desirable and for which automation would give experts a head start.

2. RELATED WORK

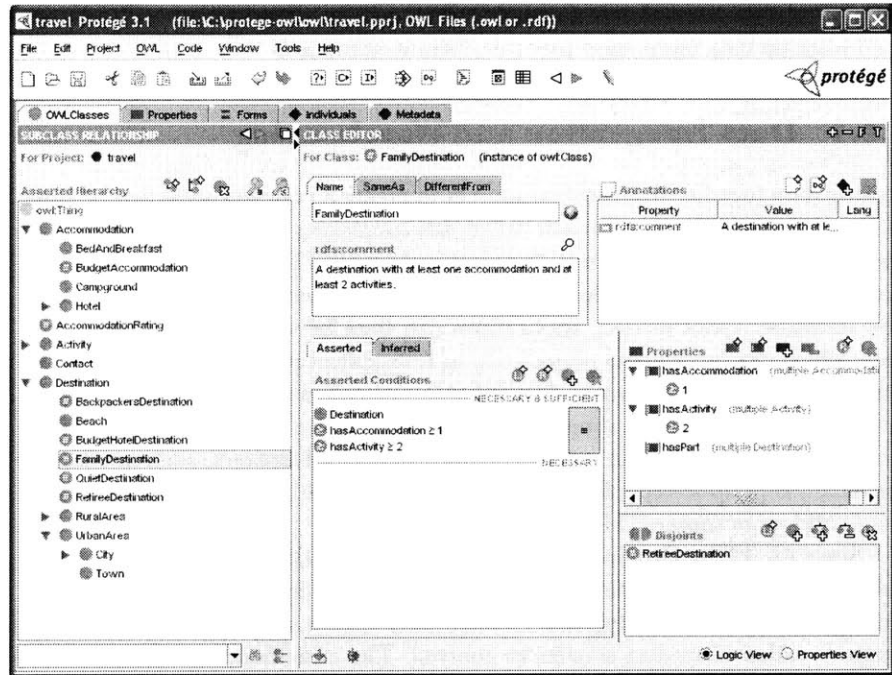


Figure 2.2. Professional ontology alignment tools, such as Protégé [23], are too advanced for casual users. They require understanding of abstract concepts like classes, class hierarchies, inheritance, inference, etc. (Image from <http://protege.stanford.edu/>.)

Ontology alignment is one specialized stage of data integration, and it demands specialized tools for ontology alignment experts. From the perspective of these experts, ontology alignment is an end, not a means. In order to do anything useful with the aligned ontologies, external tools would be needed. In contrast, for casual users ontology alignment is not an end and it is likely not even known to be a requirement.

Furthermore, these data integration tools tend to work on ontological abstractions, basing their interface interactions on concepts such as classes (Figure 2.2). Casual users have little knowledge about data modeling and ontological abstractions, and little interest in learning.

2.4.3 Data Integration User Interfaces

There has been some limited research on user interfaces for casual users to integrate data. Tabulator [43] lets casual users import RDF data from multiple sources together to create personal mash-ups with maps, calendars, timelines, etc. Not only does Tabulator consume *only* RDF data and no other format, it also provides no mechanism for aligning ontologies. It seems to make an implicit assumption that

there is a high chance of usefulness from simply pooling together data already in RDF without doing any alignment.

WebScripter [76] lets casual users create coherent reports out of data collected from several sources, offering data alignment features for that purpose. Although the desired target audience is casual users, WebScripter’s interface is still expert-oriented, full of jargon such as “DAML”, “class”, “instance”, etc. Unlike my Potluck tool, WebScripter offers no feature for fixing data at the syntactic level (e.g., swapping first name and last name) and it has not been formally evaluated on actual users.

2.5 Toward a Data-Centric Browser

As there is more and more data in reusable forms on the Web, casual users will encounter it more often and their use of the data will increase in frequency and in sophistication. Where casual users meet the Web—the web browser— will need to catch up. Designed for the text-centric Web for viewing hypertext documents, the contemporary browser may no longer be suitable for the data-centric Web. There is a need for a data-centric browser that addresses casual users’ needs in interacting with a future data-centric Web. This section surveys enhancements to the contemporary web browser that are data-centric in nature, intended for users to manage data encountered on the Web.

2.5.1 Data Models

Most of the few works aimed to enhance the standard web browser in a data-centric fashion adopt the RDF data model [28] for storing their data. They include Thresher [54] (described previously) which is built on Haystack [62]. Haystack is an RDF-based personal information management platform. The browser extension Operator [22] uses RDF as its data model and adds contextual menu commands to web page fragments marked up with microformats [17]. Tabulator [43] demonstrates what Semantic Web browsing might be like by dynamically pulling in more and more data into its client-side RDF store. The rest of the works, notably the browser extension by Dontcheva et al. [46] and Faaborg’s and Lieberman’s Creo and Miro [48], use their own data models.

2.5.2 Data-Centric Features

Four kinds of data-centric features are offered by these tools: browse data, act on data, edit data, and store data permanently. Thresher plus Haystack offer faceted browsing functionality, contextual menu commands to act on extracted data, a generic mechanism for editing, and permanent storage for the data. Dontcheva et al.’s browser extension supports permanent storage and browsing capabilities.

2. RELATED WORK

Tabulator offers only browsing features, and Operator offers mostly contextual commands. Creo and Miro are only for automation.

2.5.3 Deployment Paths

These tools can also be compared by their deployment paths. At one end, Haystack demands the full adoption of a whole new personal information management platform, of which the augmented web browser is a small component. At the other end, Tabulator is just a web page that can be loaded into any browser on demand (unfortunately, cross site scripting security restrictions on web pages prevent it from loading data from arbitrary domains). In the middle are the various browser extensions previously mentioned, which require some efforts to adopt. Less effort and less risk generally yield more adoption.

3. PUBLISHING DATA

When it came into existence, the Web was hailed for giving individuals the same publishing power as large publishers. But over time, large publishers learned to exploit the structure in their data, leveraging databases and server-side technologies to provide rich browsing and visualization features. People have come to expect from professional sites features like searching, sorting, filtering, comparison, maps, etc. Individual *small* publishers fall behind once more: neither old-fashioned static pages nor domain-specific publishing frameworks (e.g., web photo album generators) and services (e.g., Flickr) supporting limited customization can match full-fledged database-backed web applications that cost thousands of dollars. Fortunately, ideas embodied in tools that have made publishing so efficient for large publishers can also help small publishers as long as the needs and abilities of small publishers are taken into consideration. This chapter explains how.

A typical application consists of three layers: data, presentation, and application logic in between. Separating data from presentation allows mixing and matching data with presentation so that the same data can be shown in different ways. It is also easier to build tools specialized for dealing with either data (e.g., databases) or presentation (e.g., form editors) separately.

This principle of separating data from presentation has been built into technologies that target large publishers. Designed for scalability and flexibility, these technologies are far too complex for casual users to adopt. Such users have so far only been offered HTML as the generic web publishing technology, and in HTML, data and presentation are mixed together.

To let casual users benefit from the separation of data from presentation, the costs of authoring data, authoring presentation, and connecting them up must be lowered.

3. PUBLISHING DATA

The cost of authoring presentation can be lowered in two ways by assuming that the presentation needs of casual users are not so complex:

- First, a variety of common features such as sorting, grouping, searching, filtering, map visualization, timeline visualization, etc. can be provided out-of-the-box so that each casual user need not re-implement them herself. (As her presentation needs get more sophisticated and unique, she will eventually need to program her own features.)

- Second, customization to the presentation can be specified in an HTML-based syntax right inside the HTML code used to layout the web page. This is so that the publisher can work on every part of the page's presentation inside a single file in a single syntax. (As the publisher's needs to customize the presentation get more sophisticated, the HTML-based syntax will no longer satisfy her.)

The cost of authoring data can be lowered in two ways by assuming that casual users publish only small data sets:

- First, if a publisher is already editing her data in some particular format and some particular editor convenient to her, she should not need to load that data into a database and then maintain the data through the unfamiliar and inconvenient user interface of the database. She should be able to keep managing her data however she likes, and the data only gets loaded into the database when it needs to be rendered. This is realizable if the data is small and loading it into the database is quick. (As the data gets larger, there is a point when the data should already be loaded into a database to ensure a responsive web site.)

- Second, data schemas can be made optional if their benefits do not justify their costs. While schemas are advantageous on large data sets for database optimizations and for managing data at a higher level of abstraction, their benefits on small data sets are much less apparent. (As the data gets larger or more complex, there is a point when schema abstractions benefit both the publisher as well as the publishing engine.)

Finally, all the costs of setting up software (database, web server, and application server) can be eliminated if the software is packaged as a Web API to be included into a web page on-the-fly. Such a Web API can also easily allow for extensions, which accommodate the increasing sophistication in a casual user's publishing needs.

These ideas have been built into the Exhibit lightweight data publishing framework, packaged as a Web API. This chapter will discuss the design of Exhibit's user and publisher interfaces in section 1, its data model in section 2, and its user interface model in section 3. Section 4 briefly describes Exhibit's implementation. Finally, section 5 reports real-world usage of Exhibit and discusses its impact.

3.1 Interface Design

As Exhibit is a publishing framework, it has two interfaces: one facing publishers and one facing users of the published information. A web page published using Exhibit will be referred to as an *exhibit* in lowercase.

3.1.1 User Interface

An exhibit looks just like any other web page, except that it has advanced features mostly seen on commercial and institutional sites. Figure 3.1 and Figure 3.2 show two exhibits covering different types of information. Each is styled differently, but there are several common elements, as described below.

Exhibit's user interface consists mainly of *views* and *facets*, whose locations on the page are controlled by the publisher. The exhibit in Figure 3.1 is configured by the publisher to support two different views of the same data: THUMBNAILS and TIMELINE. THUMBNAILS is currently selected by the user and it is showing. The exhibit in Figure 3.2 is configured to support six views, and the BIRTH PLACES view is currently showing. Each kind of view—map, timeline, table, thumbnail, tile, scatter plot, bar chart, etc.—supports its own configuration settings. The Google Maps [11] map in the BIRTH PLACES view is configured to embed the presidents' portraits in the map marker, and the TERMS timeline view is configured to color-code the presidents' terms by their political parties. Although these settings are specified by the publisher, some can be changed dynamically by the user (e.g., sorting order in Figure 3.1). Exhibit's user interface can be extended by third-parties' views if the publisher chooses to include them.

Items can be presented differently in different views. Where there is little space to render sufficient details in-place (e.g., on a map), markers or links provide affordance for popping up bubbles containing each item's details (e.g., map bubble in Figure 3.2). The rendition of each item contains a link for bookmarking it individually. Invoking this link later will load the exhibit and pop up a rendition of the bookmarked item automatically.

The facets (left in Figure 3.1 and right in Figure 3.2) let users filter the currently displayed items. This is a conventional dynamic query interface with preview counts.

3. PUBLISHING DATA

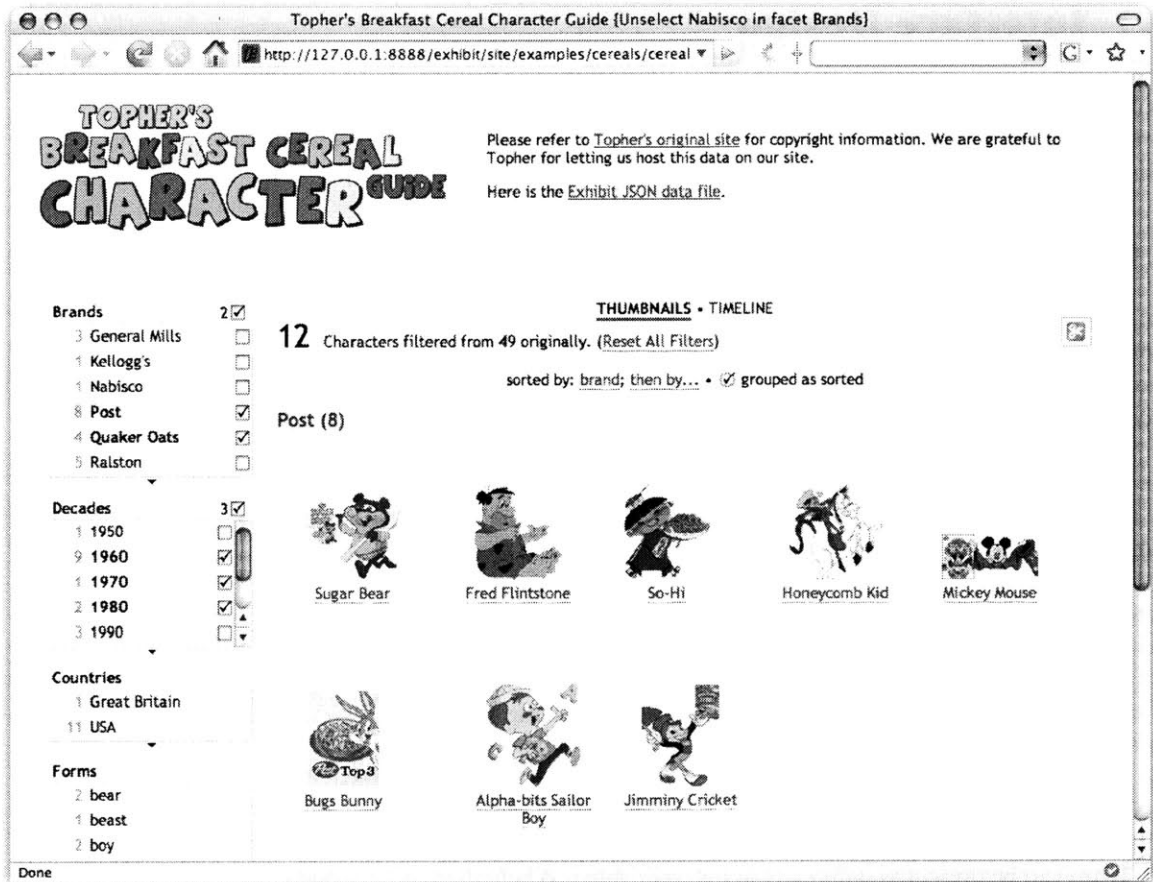


Figure 3.1. A web page embedding Exhibit to show information about breakfast cereal characters [36]. The information can be viewed as thumbnails or on a timeline and filtered through a faceted browsing interface.

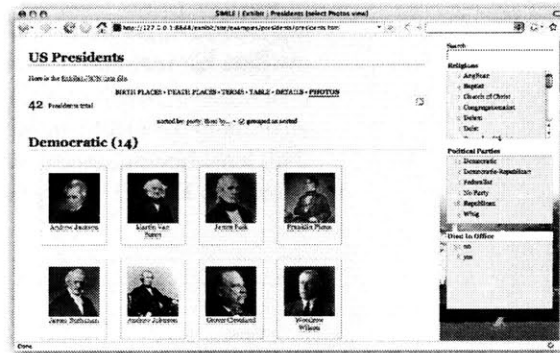
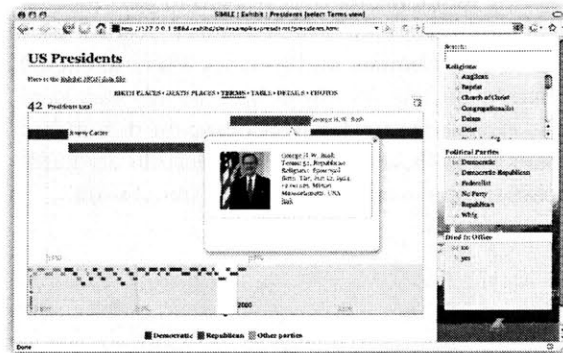
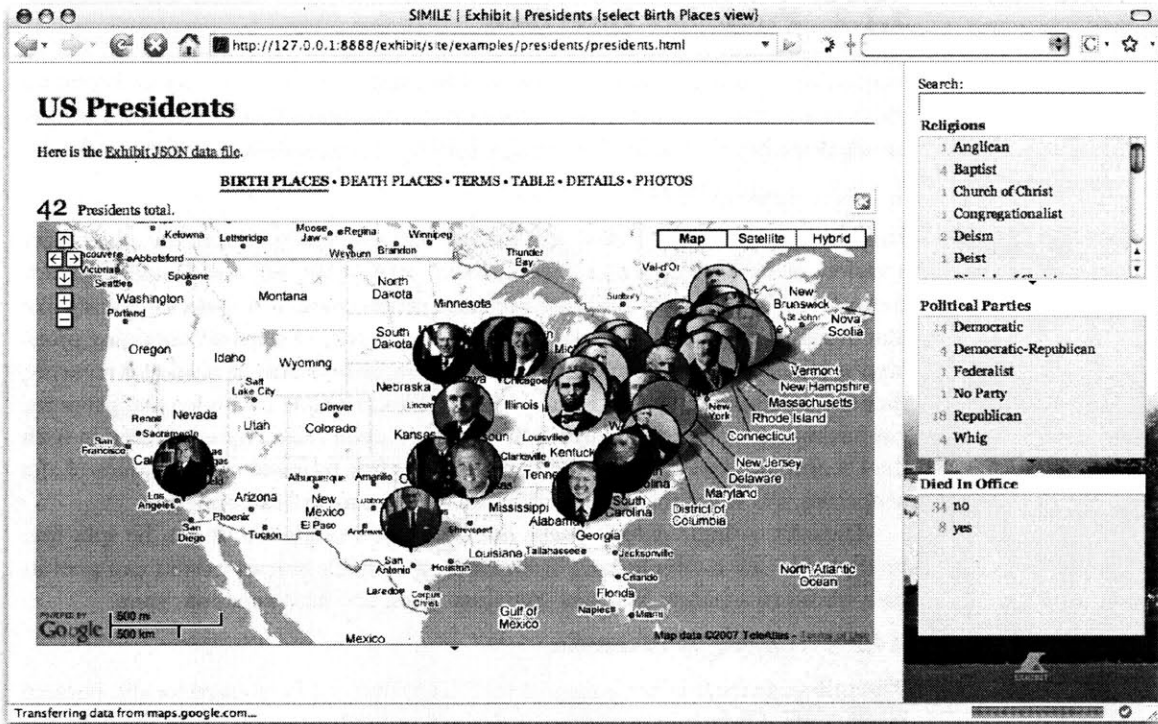


Figure 3.2. A web page embedding Exhibit to show information about U.S. presidents in 6 ways, including maps, table, thumbnails, and timelines.

3.1.2 Publisher Interface

Making an exhibit like Figure 3.1 involves two tasks: creating the data and creating the presentation. Both are iterated until the desired result is achieved. This section briefly describes the publishing process, leaving technical details to later sections.

3.1.2.1 Creating the Data

Exhibit supports its own JSON [13] format natively but can automatically import a variety of formats including Bibtex, Excel, RDF/XML, N3, and Google Spreadsheets feeds. The data file for those breakfast cereal characters looks something like that in Figure 3.3. The items' data is coded as an array of objects containing property/value pairs. Values can be strings, numbers, or booleans. If a value is an array, then the corresponding item is considered to have multiple values for that property. For instance, according to Figure 3.3, the Trix Rabbit character is released in both the US and in Canada. The publisher is mostly free to make up the names of the properties. We will discuss the specifics of the data model subsequently.

Data for a single exhibit needs not reside in a single file. It can be split into multiple files for convenience. For example, a couple's recipes exhibit can pool its data from two separate files: `her-recipes.json` and `his-recipes.json`.

3.1.2.2 Creating the Presentation

The web page itself is just a regular HTML file that can be created locally, iterated locally until satisfaction, and then, if desired, uploaded together with the data files to the web server. Figure 3.4 shows the initial HTML code needed to start making the exhibit in Figure 3.1. This code instantiates an Exhibit instance, loads it with the data file referenced by the first `<link>` element, and specifies where to embed a view panel, which shows a tile view by default. Also by default, the tile view sorts all items in the exhibit by labels and displays the top ten items using the default *lens*. This lens shows property/value pairs for each item. Reasonable defaults are hard-wired into Exhibit to give the publisher some result with minimal initial work.

```
{
  items: [
    { type:      'Character',
      label:    'Trix Rabbit',
      brand:    'General Mills',
      decade:  1960,
      country:  [ 'USA', 'Canada' ],
      thumbnail: 'images/trix-rabbit-thumb.png',
      image:    'images/trix-rabbit.png',
      text:     'First appearing on ...'
    },
    // ... more characters ...
  ]
}
```

Figure 3.3. An Exhibit JSON data file showing data for one breakfast cereal character, which is encoded as property/value pairs.

The publisher does not even need to write this initial HTML code from scratch: it is trivial to copy this code from existing exhibits or from online tutorials. This is how HTML pages are often made—by copying existing pages, removing unwanted parts, and incrementally improving until satisfaction. The declarative syntax of HTML, the forgiving nature of web browsers and their reasonable defaults, and the quick HTML edit/test cycle make HTML publishing easy and instantly gratifying. Exhibit has been designed to afford the same behavior.

Figure 3.5 shows the final HTML code needed to render the exhibit in Figure 3.1 (logo graphics and copyright message omitted). The additional code, in black, configures the facets, the two views, and a lens template in the THUMBNAILS view. (Lens templates will be discussed in the User Interface Model section.)

Making the presentation look better can also involve filling in and fixing up the data schema. Figure 3.6 shows how the plural label for the type **Character** is declared so that plural labels in the UI, e.g., **12 Characters**, can be generated properly. The **decade** property values are declared to be dates instead of strings so that they can be sorted as dates.

To change the schema, e.g., renaming a property, the publisher can simply invoke the text editor's Replace All command. Or if the data is imported from a spreadsheet, she can just rename the corresponding column header label. Saving old versions of the data involves making copies of the data files. Changing schema and keeping versions of the data might not be as simple if databases were used.

Thus, just by editing one or two text files in any text editor, and perhaps editing data in a spreadsheet, a casual user with only basic knowledge of HTML and no programming skills can create a richly interactive web page with sorting, searching, filtering, maps, timelines, etc.—features that would otherwise take a whole team of web engineers months to implement.

```
<html>
<head>
  <title>Topher's Breakfast Cereal Character Guide</title>
  <link type="text/javascript"
    rel="exhibit/data" href="cereal-characters.js" />
  <script type="text/javascript"
    src="http://static.simile.mit.edu/exhibit/api-2.0/exhibit-api.js">
  </script>
</head>
<body>
  <div ex:role="viewPanel"></div>
</body>
</html>
```

Figure 3.4. To create the web page in Figure 3.1, the author starts with this boiler plate HTML code, which displays the characters in `cereal-characters.js` through the default lens that lists property/value pairs.

3. PUBLISHING DATA

```
1 <html>
2 <head>
3   <title>Topher's Breakfast Cereal Character Guide</title>
4   <link type="application/json" rel="exhibit/data" href="cereal-characters.json" />
5   <script src="http://static.simile.mit.edu/exhibit/api-2.0/exhibit-api.js"></script>
6   <script src="http://static.simile.mit.edu/exhibit/extensions-2.0/
7     time/time-extension.js"></script>
8   <style>
9     .itemThumbnail {
10      width: 120px;
11    }
12  </style>
13 </head>
14 <body>
15   <table width="100%">
16     <tr valign="top">
17       <td width="20%">
18         <div ex:role="facet" ex:expression=".brand" ex:facetLabel="Brands"></div>
19         <div ex:role="facet" ex:expression=".decade" ex:facetLabel="Decades"></div>
20         <div ex:role="facet" ex:expression=".country" ex:facetLabel="Countries"></div>
21         <div ex:role="facet" ex:expression=".form" ex:facetLabel="Forms"></div>
22       </td>
23       <td>
24         <div ex:role="viewPanel">
25
26           <div ex:role="view" ex:viewClass="Thumbnail"
27             ex:possibleOrders=".brand, .decade, .form, .country">
28
29             <div ex:role="lens" class="itemThumbnail">
30               <img ex:src-content=".thumbnail" />
31               <div ex:content="value"></div>
32             </div>
33           </div>
34
35           <div ex:role="view"
36             ex:viewClass="Timeline"
37             ex:start=".year"
38             ex:colorKey=".topic">
39           </div>
40         </div>
41       </td>
42     </tr>
43   </table>
44 </body>
45 </html>
```

One or more links to data

Exhibit API and extensions

custom styles

facets

lens templates specifying how to render each data item

views

Figure 3.5. The publisher starts with the code in gray (from Figure 3.4), includes more and more of the code in black, and tweaks until the desired result (Figure 3.1) is achieved (logo graphics and copyright omitted). Tweaking involves following online documentation or just copying code from other existing exhibits.

```

{
  types: {
    'Character': { pluralLabel: 'Characters' }
  },
  properties: {
    'url': { valueType: "url" },
    'decade': { valueType: "date" }
  }
  items: [
    // ... items ...
  ]
}

```

Figure 3.6. Schema information can be added to the JSON file to improve Exhibit’s user interface.

3.2 Data Model

An Exhibit data model is a directed graph in which the nodes are either *items* or native values such as numbers, strings, and booleans, and the arrows are *properties* (Figure 3.7). Each property has a property value type which specifies the types—numbers, strings, booleans, items, etc.—of the nodes at the pointed ends of the arrows. In the simplest case where there is no relationship between items (no blue curved arrows in Figure 3.7), then the data model degenerates into a flat list of items with native-valued properties. A casual user would normally start with such a conceptually simple list of items and then when the need actually arises, link the items up to form a graph.

3.2.1 Items

Each item has one or more properties, one of which is the mandatory **label** property, which is a string naming that item in a human-friendly manner (friendliness is subject to the publisher’s discretion). This label is used to render the item whenever a concise, textual description of that item is needed.

Every item is also mandated to have an **id** property, which identifies that item uniquely within the containing exhibit. If no **id** property value is specified explicitly when the item is being loaded into the database of the exhibit, the item’s **label** property value is used as its **id**. Other items in the same exhibit can relate to this item simply by having property values equal to the item’s **id**.

The third mandated property is **uri**, which is the URI used to name the item when it is exported to any RDF [28] serialization format. (The **uri** property helps make exhibits’ data readily available to semantic web applications.) If no **uri** property value is specified when the item is being loaded, the URI is generated automatically by appending its **id** property value to the URL of the containing exhibit.

The last mandated property is **type**. An item’s **type** property value is just a string, such as “President” and “Publication”. If not explicitly specified, the item’s

3. PUBLISHING DATA

type defaults to “Item”. Types are introduced to divide the set of items within a single exhibit into several subsets of conceptually different items, such as “Publication” vs. “Author”, or “Patient” vs. “Hospital Record”. If the data within an exhibit were to be stored in a relational database, there would logically be one relational table for each type.

Although there are four mandated properties, a publisher is only burdened to make up one—the **label** property—for each item. Note that in Figure 3.3, the item has neither **id** nor **uri** property value; both values will be generated. An **id** property value must be specified explicitly if another item with the same label has already been loaded into the database. A **uri** property value must be specified explicitly if the publisher intends the item to refer to some Web resource with an existing URI. For example, in an exhibit that compares several web sites’ traffic on a bar chart, each item corresponds to one web site and the item’s **uri** should logically be the web site’s URL.

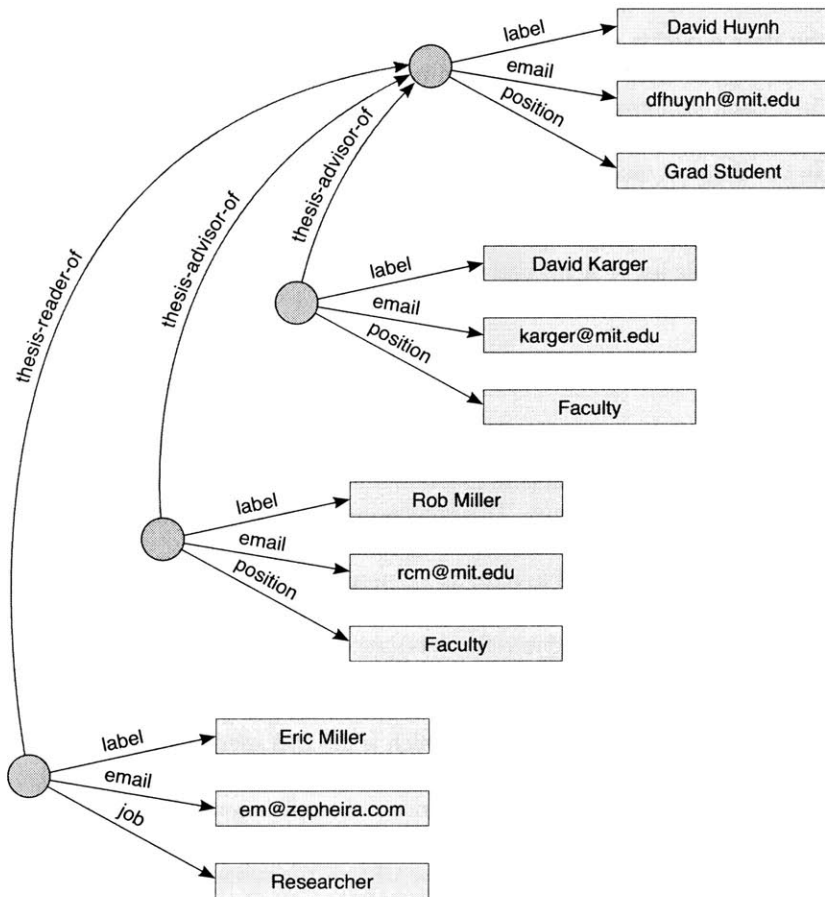


Figure 3.7. An Exhibit data model is a graph in which the nodes are items (circles) or native values (rectangles) and the arrows are properties.

3.2.2 Types

The information on types is schematic information. Each type has three mandated schematic properties: **label**, **id**, and **uri**. Note that when the **type** property of an item is specified, that **type** property value is the type's **id**, not the type's **label** (Figure 3.8). Whereas usually an item's **label** is specified and its **id** is generated from its **label**, a type's **id** must be specified first (by assigning some item that **type** property value) and then the type's **label** is taken to be the same as its **id**, unless overridden. Whereas for items, **ids** are primarily used to distinguish those with the same **label**, for types, **ids** are used as short-hand notations. For example, in Figure 3.8, the types' **ids** save the publisher a lot of typing while the types' **labels** are easy to comprehend for viewers.

Beside **id**, **uri**, and **label**, a type can have other schematic properties that help in the localization of the user interface. For instance, when a type has a **pluralLabel**, that schematic property value is used in English-speaking locales to generate user interface text for labeling several things of that type, e.g., **9 People** instead of **9 Person**.

There is no need for the publisher to explicitly declare every type in the **types** section. A type is added to the system whenever an item of that type is added. This lets the publisher focus on the items—the main point of her exhibit—and only add information on types and properties when they make the user interface better.

```

{
  items: [
    ...
    { ...
      type: "RCAct",
      ...
    },
    ...
  ],
  types: {
    "RCAct": {
      label: "Ribosomal Chaperone Activity",
      pluralLabel: "Ribosomal Chaperone Activities",
      uri: "http://www.geneontology.org/go#GO:0000005"
    },
    "RMR": {
      label: "Regulation of Mitotic Recombination",
      pluralLabel: "Regulations of Mitotic Recombination",
      uri: "http://www.geneontology.org/go#GO:0000019"
    }
  }
}

```

a type's id

Figure 3.8. The **types** section in the JSON file specifies schematic properties of types, such as their **labels** and **uris**.

3.2.3 Properties

Schematic information can also be given on properties. Properties have the same three mandated schematic properties as types (`label`, `id`, and `uri`) plus two more: `reverseLabel` and `valueType`. Together with `label`, `reverseLabel` lets a property be described in both directions (e.g., `child of` in one direction and `parent of` in the other).

A property's `valueType` hints at the nature of that property's values—whether they are numbers, or booleans, or strings, or items' `ids`, etc. There is a fixed set of value types, currently including: `text`, `number`, `date`, `boolean`, `url`, and `item`. All values of type `item` are locally unique `ids` of items in the same exhibit. Figure 3.6 shows how property value types are declared in data files.

Note that such declaration of value types are hints only. Exhibit will try to coerce property values into whatever types that are most appropriate for each particular use. For example, when a property `foo` with value type `text` is used to specify the starting dates of items on a timeline view, then all `foo` property values are automatically parsed into dates to render the timeline.

This design choice of tying value types to properties, rather than requiring type declaration on each value, facilitates incremental improvements to the data. For instance, `brand` property values in Figure 3.3 are initially, by default, of type `text`. This might satisfy the publisher until she wants to record details about the brands, at which time she can simply specify the value type for `brand` to be `item` and add the data for the brands (Figure 3.9).

3.2.4 Expressions

Exhibit provides an expression language that makes it convenient to access its graph-based data model.

Consider an exhibit whose data model is illustrated in Figure 3.10. The exhibit contains information on some publications, their authors, the schools where the authors teach, the countries where the schools are located, and perhaps more. Evaluating the expression `.written-by` on some publications yields their authors; evaluating `.written-by.teaches-at` yields the schools where the publications' authors teach; and so forth. The *hop operator* `.` traverses the graph *along* a given property. To traverse *against* a given property, the hop operator `!` is used. Thus, evaluating `!teaches-at!written-by` on some schools returns publications written by authors at those schools. The two kinds of hop operator can be mixed: evaluating `.written-by.teaches-at!teaches-at!written-by` on some publications returns all publications written by all authors teaching at the same schools as the authors of the original publications.

A sequence of alternating hop operators and property `ids`, such as `.written-by.teaches-at`, is called a *path*. Evaluating a path yields a collection of values (with or without duplicates). Why duplicate values are allowed will become clear subsequently.

```

{
  properties: {
    'brand': {
      valueType: 'item'
    }
  },
  items: [
    { type: 'Character',
      label: 'Trix Rabbit',
      brand: 'General Mills',
      decade: 1960,
      country: [ 'USA', 'Canada' ],
      thumbnail: 'images/trix-rabbit.png',
      text: 'First appearing on ...'
    },
    // ... more characters ...

    { type: 'Brand',
      label: 'General Mills',
      headQuarter: 'Minnesota',
      founded: 1856
    },
    // ... more brands ...
  ]
}

```

Figure 3.9. Elaboration of brand property values by specifying the value type for the property **brand** and adding Brand items.

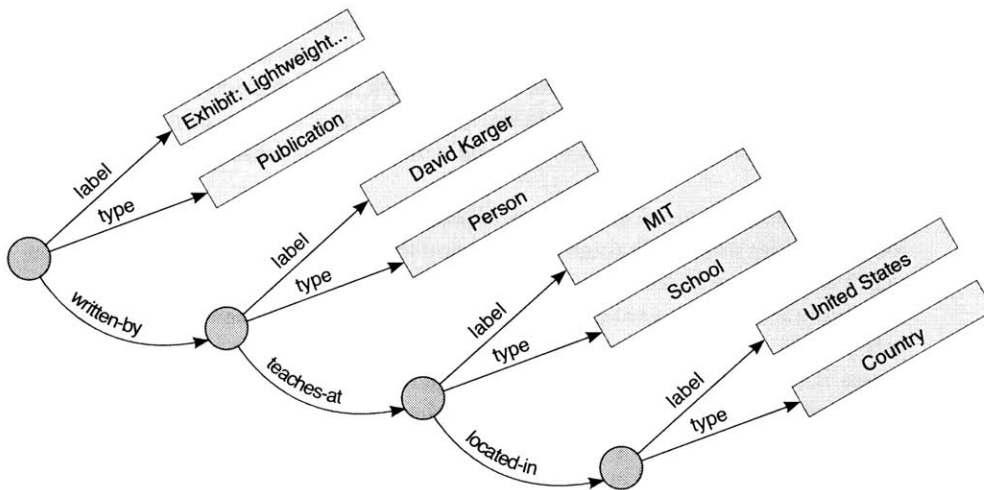


Figure 3.10. In this sample Exhibit data model, evaluating the expression `.written-by.teaches-at.located-in` on publications yields the countries in which the schools where the publications' authors teach are located.

3. PUBLISHING DATA

In addition to paths, Exhibit expressions can also contain native values like strings and numbers, and *functions*, *operators* and *constructs*. For example, `concat(.last-name, ' ', .first-name)` computes a full name out of a last name and a first name, and `count(.parent-of)` returns the number of children of a person. Other functions include:

- **union** (for taking set unions of collections);
- **contains** (for testing if a collection contains a given element);
- **exists** (for testing if another expression returns any result, that is, if the result collection is non-empty);
- **add** and **multiply** (for taking summations and products of elements in one or more collections);
- **and**, **or**, and **not** (for combining boolean values);
- **date-range** for computing differences between two dates in a particular unit such as in days or in years;
- **distance** for computing geographical distance between two latitude/longitude pairs.

It is easy for third parties to implement more functions. Publishers just need to link the third parties' Javascript code into their exhibits.

Exhibit's operators are the standard arithmetic operators (+, -, *, /) and boolean operators (=, <>, >, <, <=, >=).

There are two constructs at the moment:

- **if**, which evaluates its first argument to a boolean value and based on that, chooses to evaluate either its second argument or its third argument;
- **foreach**, which evaluates its first argument into a collection; then for each element in that set, evaluates the expression that is its second argument on that element; and finally returns the union of all results.

The reason we need expressions to support duplicates in collections is as follows. Consider an exhibit about students and their assignments. To compute a student's average, we divide the sum of the grades of her assignments by the number of assignments: `add(.assignment.grade)/count(.assignment)`. Say a student gets 100 for two assignments. If the result of the sub-expression `.assignment.grade` is a collection without duplicates, then the student's average is computed incorrectly as $100 / 2 = 50$. To fix this problem, Exhibit introduces two more hop operators `!@` and `@` that prevent duplicates from being eliminated further down the evaluation process. Thus, `add(.assignment.@grade)/count(.assignment)` would compute the correct average.

The full production rules for Exhibit expression language is provided in Figure 3.11.

3.2.5 Data Import/Export

While the Exhibit JSON format is relatively readable and writeable, it might not be the format of choice for some publishers. They might already have their data in another format (e.g., BibTex), or they might be used to editing tools that do not export to Exhibit JSON (e.g., Excel). These publishers can simply annotate the `<link>` elements referencing their data with the right mime-type (e.g., `type="application/bibtex"`) and Exhibit will import it automatically. Exhibit's importer architecture allows for third parties to plug in their own importers.

Exhibit also has an extensible exporter architecture that can serialize its data into several formats, ready to be copied off. The natively provided exporters can generate RDF/XML, Exhibit JSON, Semantic MediaWiki extension wikitext [73], and Bibtex. The purpose of these exporters is to facilitate and encourage propagation of reusable data by offering convenience to both users and publishers. For example, being able to copy off the Bibtex of some publications that you have found in an exhibit so that you can cite them is very convenient. You can also copy that

```

<expression>      ::= <sub-expression>
                  | <expression> <expr-op> <sub-expression>
<sub-expression> ::= <term>
                  | <sub-expression> <subexpr-op> <term>
<term>           ::= <factor>
                  | <term> <term-op> <factor>
<factor>        ::= <number>
                  | <string>
                  | <construct>
                  | <function-call>
                  | <path>
                  | "(" <expression> ")"

<construct>      ::= <if-construct>
                  | <foreach-construct>
                  | <default-construct>
<if-construct>   ::= "if" "(" <expression> "," <expression> "," <expression> ")"
<foreach-construct> ::= "foreach" "(" <expression> "," <expression> ")"
<default-construct> ::= "default" "(" <expression-list> ")"

<function-call> ::= <identifier> "(" <expression-list?> ")"

<expression-list> ::= <expression>
                  | <expression-list> "," <expression>

<path>          ::= <hop-list>
                  | <identifier> <hop-list>
<hop-list>      ::= <hop>
                  | <hop-list> <hop>
<hop>          ::= <hop-op> <property-id>

<expr-op>       ::= "=" | "<" | "<" | "<=" | ">" | ">="
<subexpr-op>    ::= "+" | "-"
<term-op>       ::= "*" | "/"
<hop-op>        ::= "." | "!" | ".@" | "!"@

```

Figure 3.11. Production rules for the Exhibit expression language.

data in Exhibit JSON format and incorporate it into your own exhibit to make an archive of related work.

Exhibit's default exporters generate an **origin** property value for each item to export. This value is the URL that, when invoked, returns to the original exhibit and pops up the view of that item automatically. This is a lightweight mechanism for attribution.

3.3 User Interface Model

Exhibit's user interface consists of several types of component:

- *collections*;
- widgets: *facets*, *views*, and *lenses*; and
- helpers: *coders*.

Exhibit also maintains a hierarchy of *UI contexts* that store inheritable UI settings such as *formatting rules*. These components and the formatting rules can all be specified in HTML. The following subsections define these various UI concepts and describe how they can be specified in HTML.

3.3.1 Collections

A collection contains a subset of the items currently in the exhibit's database. Currently, a collection can be defined to contain items by some particular types, e.g.,

```
<div ex:role="collection" ex:itemTypes="Man; Woman" id="adults"></div>
<div ex:role="collection" ex:itemTypes="Boy; Girl" id="kids"></div>
<div ex:role="collection" ex:itemTypes="Man; Woman; Boy; Girl"></div>
```

Each collection has an ID, which can be specified using the HTML **id** attribute. The ID of a collection is used to refer to it from other components, as will be discussed in later subsections. If no **id** attribute is given, the ID is assumed to be "**default**" and the effect of the definition is to redefine the **default** collection, which comes, by default, with every exhibit. If not redefined, the **default** collection contains all items currently in the exhibit's database.

What a collection originally contains is called its *root set*. This root set can be filtered down to a *filtered set* by facets attached to the collection, as will be discussed next.

3.3.2 Facets

A facet is a component whose purpose is to filter a collection's root set down to a filtered set. Facets can be declared as follows:

```
<div ex:role="facet"
  ex:expression=".ethnicity"
></div>
```

```

<div ex:role="facet"
  ex:expression=".profession"
  ex:collectionID="adults"
></div>
<div ex:role="facet"
  ex:expression=".height * 2.54"
  ex:facetLabel="Height (cm)"
  ex:collectionID="kids"
  ex:facetClass="NumericRange"
  ex:interval="10"
></div>

```

The `ex:expression` attribute is required and in the simplest case, it specifies the property by which to filter the collection to which the facet is attached. In the first example, the facet can be used to filter the attached collection by (people's) ethnicity; in the second, by professions; and in the third, by heights. The expression does not have to be a simple property; it can compute more complex values such as seen in the third example. The `ex:collectionID` specifies which collection a facet is attached to; if missing, the `default` collection is used (as in the first example).

The `ex:facetLabel` attribute gives a text label to the facet. If missing, and if the expression is a simple property, Exhibit uses the property's label; otherwise, Exhibit shows an error message where the label should be rendered.

The `ex:facetClass` attribute specifies which class of facet to instantiate. There are two classes being supported: `List` and `NumericRange` (`List` is the default class). `List` facets show each value computed by the expression as a choice for filtering while `NumericRange` facets convert each value into a number and group values into intervals.

3.3.3 Views

A view is a component that renders the filtered set of items in a collection to which it is attached. As the collection gets filtered by the facets attached to it, the views attached to that collection update themselves to show the current filtered set.

Views can be specified much like facets. Each kind of view supports its own settings that are appropriate to it. For example, a tabular view supports settings pertaining to its columns, e.g.,

```

<div ex:role="view" ex:viewClass="Tabular"
  ex:columns      = ".label, .imageUrl, .party, .age"
  ex:columnFormats = "text, image, text, number"
  ex:sortColumn   = "2"
  ex:sortAscending = "true"
></div>

```

3.3.4 Lenses

An Exhibit lens renders one single item. Lenses are *not* specified individually per item, but they are instantiated from a few *lens templates*, which are specified in HTML.

A template is just a fragment of HTML that can be specified in-line, as in Figure 3.12, or in a different file.

Within a lens template, the **content** attribute of an element specifies what content to stuff into that element when the template is instantiated for an item. For example, in Figure 3.12, the **<h1>** element will be filled with the **label** property value of the item.

Attributes that end with **-content** are assumed to contain Exhibit expressions. These expressions are resolved into actual values when the lens is instantiated, and the values are used to assert HTML attributes of the same name but without the **ex:** namespace and the **-content** suffix. For example, the **ex:src-content** attribute in Figure 3.12 is replaced with the **image** property value of the item being rendered, generating the attribute **src="images/trix-rabbit.png"** in the generated HTML **** element.

The **ex:if-exists** attribute of an element determines whether that element and its children in the template should be included in the presentation of a particular item. For example, if an item does not have an **image** property value, the template in Figure 3.12 will not generate a broken image. The **ex:if** attribute of an element specifies an expression that evaluates to a boolean; if the boolean is true, the first child element of the element is processed; otherwise, the second child element is processed if it is present. The **ex:select** attribute of an element specifies an expression that evaluates to some value which is used to select which child element to process. The child element with the matching **ex:case** attribute value is processed. Otherwise, the last child element that does not have any **ex:case** attribute is processed.

The **ex:control** attribute specifies which Exhibit-specific control to embed. There are only two controls supported at the moment: the **item-link** control and the **copy-button** control; the first is a permanent link to the item being rendered and the latter is a drop-down menu button that lets the user copy the item's data off in various formats.

3.3.5 UI Contexts and Formatting Rules

Each component has a UI context which gives it access to UI settings, such as how dates should be formatted (e.g., "July 4" or "7/4" or "7月4日"). Such formatting rules can be added to the UI context of a component through the **ex:formats** attribute on the component's HTML specification, e.g.,

```
<div ex:role="view"
  ex:viewClass="Timeline"
  ex:start=".birth"
  ex:formats=
    "date { mode: short; show: date } number { decimal-digits: 2 }"
></div>
```

In this example, whenever dates are rendered inside that timeline view, they are displayed in short form without time of the day (e.g., "04/07/07" for July 4, 2007). Whenever numbers are displayed, two decimal digits are shown.

```

<table ex:role="exhibit-lens" cellpadding="5" style="display: none;">
<tr>
<td>
<img ex:if-exists=".cereal" ex:src-content=".image" />
<div ex:control="copy-button"></div>
</td>
<td>
<h1 ex:content=".label"></h1>
<h2>
<span ex:content=".brand"></span>
<span ex:content=".cereal"></span>
</h2>
<p ex:content=".text"></p>
<center ex:if-exists=".url">
<a ex:href-content=".url" target="new">More...</a>
</center>
</td>
</tr>
</table>

```

Figure 3.12. Lens template for showing a breakfast cereal character in a pop-up bubble as shown in Figure 3.13.

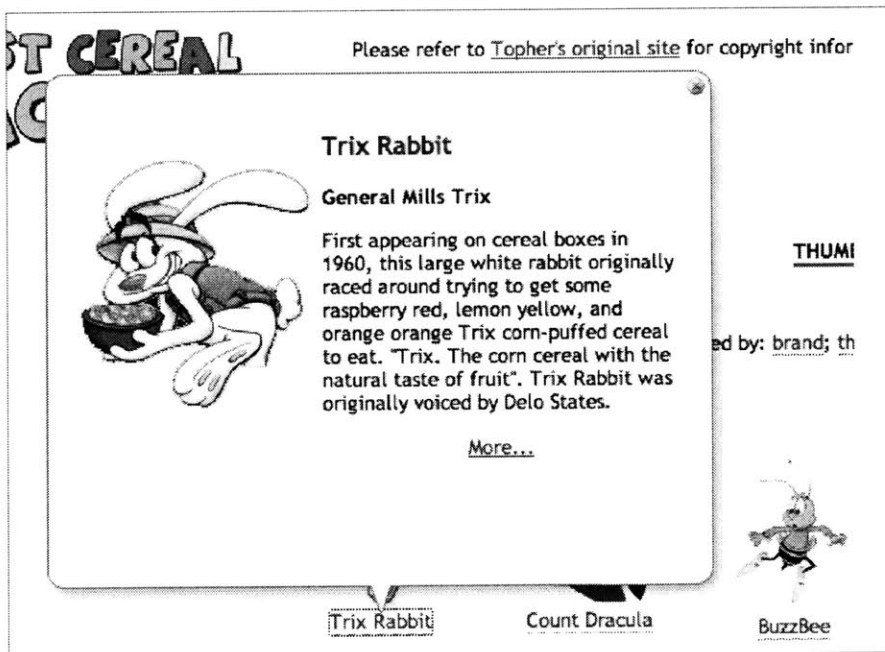


Figure 3.13. Bubble showing a breakfast cereal character.

3. PUBLISHING DATA

Note that the resemblance in syntax to CSS is intended for familiarity. An **ex:formats** specification consists of zero or more *rules* (cf. CSS rules). Each rule consists of a selector which specifies what the rule applies to (cf. CSS selectors) and then zero or more *settings* inside a pair of braces. In the example, there are two rules and two selectors, **date** and **number**. Those two selectors select the value types to which the rules apply.

For each selector there is a different set of allowable settings. For example, for **date** there are **time-zone**, **mode**, and **template**. For each setting there is a set of acceptable setting values. For example,

- **time-zone** takes a number or the keyword **default**;
- **mode** takes any one of the keywords **short**, **medium**, **long**, and **full** to specify how detailed to render dates and times;
- **template** takes a string that acts as a template for formatting dates and times in case those four standard modes do not satisfy the publisher's needs; and
- **negative-format** (for currency) takes any combination of the flags **red**, **black**, **parentheses**, **no-parentheses**, **signed**, and **unsigned** to specify how negative currency values should be rendered.

As in CSS, URLs in formatting rules are wrapped in **url()**. Similarly, expressions are wrapped in **expression()**. For example,

```
ex:formats=  
  "item { title: expression(concat(.full-name, ', ', .job-title)) }"
```

specifies that when a textual description of an item is needed, render the **full-name** property value followed by a comma and the **job-title** property value.

The full production rules for Exhibit's format language is provided in Figure 3.14.

Other UI settings beside formatting rules can be specified through other attributes, such as **ex:bubbleWidth** for the width in pixels of popup bubbles. Settings not specified in a view's HTML specification are inherited from the view's outer

<rule-list>	::= <rule>*
<rule>	::= <selector> ["{" [<setting-list>] }"
<selector>	::= "number" "date" "boolean" "text" "image" "url" "item" "currency" "list"
<setting-list>	::= <setting> <setting-list> ";" <setting>
<setting>	::= <setting-name> ":" <setting-value>
<setting-value>	::= <number> <integer> <non-negative-integer> <string> <keyword> <url> <expression> <flags>
<flags>	::= <keyword>+

Figure 3.14. Production rules for the Exhibit format language.

lexical scope, that is, from the settings specified on components whose HTML specifications lexically contain this view’s HTML specification.

3.3.6 Coders

A coder translates a piece of information to some visual feature, e.g.,

- from the `political-party` property value “Republican” to the color red, and from “Democrat” to the color blue;
- from the `service` property value “Restaurant” to an icon showing a fork and a knife, and from “Hotel” to an icon showing a bed;
- from the `magnitude` property value 5 (assumed to be the magnitude of an earthquake) to the number 40, indicating how large the corresponding map marker should be;
- from the `temperature` property value -30 (assumed to be in Celsius) to the color code `#0000aa`, a particular shade of blue indicating how cold it is.

For the four examples above, these are the coders’ specifications:

```
<div ex:role="coder" ex:coderClass="Color" id="political-party-colors">
  <span ex:color="red">Republican</span>
  <span ex:color="blue">Democrat</span>
  <span ex:color="gray" ex:case="default">Any other party</span>
  <span ex:color="white" ex:case="mixed">Multiple parties</span>
  <span ex:color="black" ex:case="missing">No party</span>
</div>

<div ex:role="coder" ex:coderClass="Icon" id="service-icons">
  <span ex:icon="fork-knife.png">Restaurant</span>
  <span ex:icon="bed.png">Hotel</span>

  <span ex:icon="question-mark.png"
    ex:case="default">Other service</span>
  <span ex:icon="many-question-marks.png"
    ex:case="mixed">Multiple services</span>
  <span ex:icon="x.png"
    ex:case="missing">No service</span>
</div>

<div ex:role="coder"
  ex:coderClass="NumericGradient" id="earthquake-magnitudes"
  ex:gradientPoints="1, 20; 10, 60"
  ></div>

<div ex:role="coder" ex:coderClass="ColorGradient" id="temperature-col-
ors"
  ex:gradientPoints="-40, #000088; 0, #ffffff; 50, #ff0000"
  ></div>
```

The `ex:case` attribute is used to specify special cases, such as when a single marker on the map corresponds to several values to translate (e.g., because several politicians belong to different political parties were born in the same city), or when there is no data to translate (e.g., because there is no temperature recorded for a particular city). The `ex:case="default"` attribute value is used to code all remaining cases (e.g., other political parties beside Democrat and Republican).

3. PUBLISHING DATA

To connect a view or a facet to a coder, you need to specify on the view or facet's HTML specification which coder to use and what data to feed it. To have a map view plotting politicians colored by political parties, then we can specify that map view as follows:

```
<div ex:role="view" ex:viewClass="Map"
    ex:latlng=".latlng"
    ex:colorCoder="political-party-colors"
    ex:colorKey=".party"
></div>
```

As another example, to construct a map view of buildings and the services available in them, write:

```
<div ex:role="view" ex:viewClass="Map"
    ex:latlng=".latlng"
    ex:iconCoder="service-icons"
    ex:iconKey=".service"
></div>
```

Several coders could potentially be used in the same view. For example, on a map plotting natural disasters, an icon coder can indicate whether a disaster is an earthquake, a volcano eruption, a tsunami, etc.; a marker size coder can show the magnitudes of those disasters; and a color coder can show the casualties (e.g., more red means more casualties).

3.4 Implementation

The Exhibit framework is implemented in several Javascript, CSS, and image files. It is available at a public URL where anyone can reference it from within his or her HTML pages. Exhibit publishers do not need to download any software, and users who view exhibits do not need to install any browser extension. This zero cost is the signature of client-side Web APIs and is largely responsible for the explosion in the embedding use of Google Maps [11].

Exhibit's source code is available publicly. Any of its parts can be overridden by writing more Javascript code and CSS definitions after including Exhibit's code. Third parties can implement additional components to supplement Exhibit.

Exhibit's architecture is illustrated in Figure 3.15. At the bottom is the data layer consisting of the database, the expression language parser and evaluator, and importers and exporters. At the top is the user interface layer, which consists of three sub-layers:

- UI contexts and localization resources—storage of presentation settings for the rest of the user interface layer.
- collections and coders—components that do not render to the screen but determine what data widgets should render and how to render it.
- widgets which perform the actual rendering and support interactions.

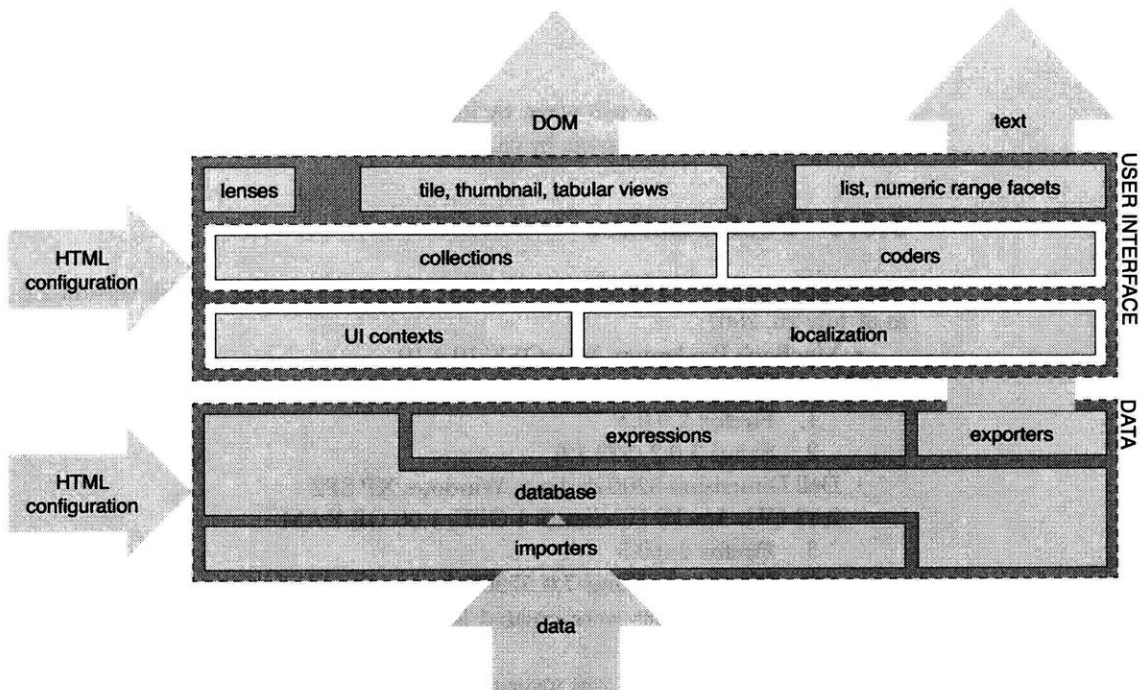


Figure 3.15. Exhibit’s architecture

There are several points of extensibility. More views, facets, and coders can be added. More importers, exporters, and functions (to be used in expressions) can be registered. For example, a new facet class can be added to show a calendar or a timeline supporting date range selection instead of just listing individual dates.

The localization component encapsulates localized UI resources, including text strings, images, styles, and even layouts. This is only our early attempt—internationalizing a framework that generates user interfaces at run-time is very difficult. We note that even HTML is biased for English. For example, bold and italics, which have native syntax in HTML, are foreign concepts to most Asian scripts.

3.5 Evaluation

Exhibit was evaluated in two ways: by its performance in contemporary popular browsers and by its actual usage by casual users.

3.5.1 Performance

Four platform/browser configurations were chosen for testing Exhibit version 2.0 as of July 26, 2007:

- MacBook Pro laptop, Mac OSX 10.4.10
2.16 GHz Intel® Core 2 Duo, 2 GB RAM
 1. Firefox 2.0.0.5
 2. Safari 3.0.2 (522.12)
- Dell Dimension 8200 desktop, Windows XP SP2
2.53 GHz Intel® Pentium® 4 CPU, 1.00 GB RAM
 3. Firefox 2.0.0.5
 4. Internet Explorer 7.0.5730.11

For Firefox, new profiles were created for testing. For the other browsers, their caches were cleared.

Exhibit's performance was measured in two ways: by its load time and by its interactivity time. The only independent variable was the number of items in the test exhibit, which varied from 500 to 2,500. The results are shown in Figure 3.16 and Figure 3.17, which will be explained below.

3.5.1.1 Load Time

Load time was divided into two stages:

- Data loading: populating the database with data that has already been received at the browser, thus excluding network traffic cost;
- UI rendering: constructing the initial presentation of the exhibit, which includes all the costs of computing the facets, querying the database for data to construct the lenses, and actually generating DOM elements.

There were two variations of UI rendering:

- All items were sorted by label and rendered in a tile view.
- All items were sorted by label but only the first 10 were rendered.

Four platform/browser configurations combined with two variations of UI rendering yielded the eight charts shown in Figure 3.16. These charts show that UI rendering cost considerably more than data loading, and that by rendering only the first 10 items, reasonable load time performance (under 5 seconds) could be achieved in all tested platform/browser configurations for 1,000 item exhibits.

3.5.1.2 Interactivity Time

To test interactivity time, I measured the time it took from clicking on a facet value with a count of 3 (after the test exhibit has finished loading and rendering) to when

the exhibit has finished rendering the filtered set of items and updating the facets. There were also two variations of initial UI rendering as in the load time test, producing a total of eight traces in the two plots in Figure 3.17. Rendering all items at the beginning did affect the filtering time substantially, perhaps because there was a high cost for removing generated DOM elements. Responsive performance (within half a second) could be achieved in all platform/browser configurations if only 10 items were rendered initially.

These test results yielded encouraging evidence that reasonable performance is achievable by Exhibit's client-side approach. More browser optimizations and faster hardware will soon make Exhibit scale better. Already we see that the latest beta version of Safari double the speed of the latest released version of Firefox on the same platform.

3. PUBLISHING DATA

Mac OSX 10.4.10, 2.16 GHz Intel Core 2 Duo, 2 GB 667 DDR2 SDRAM

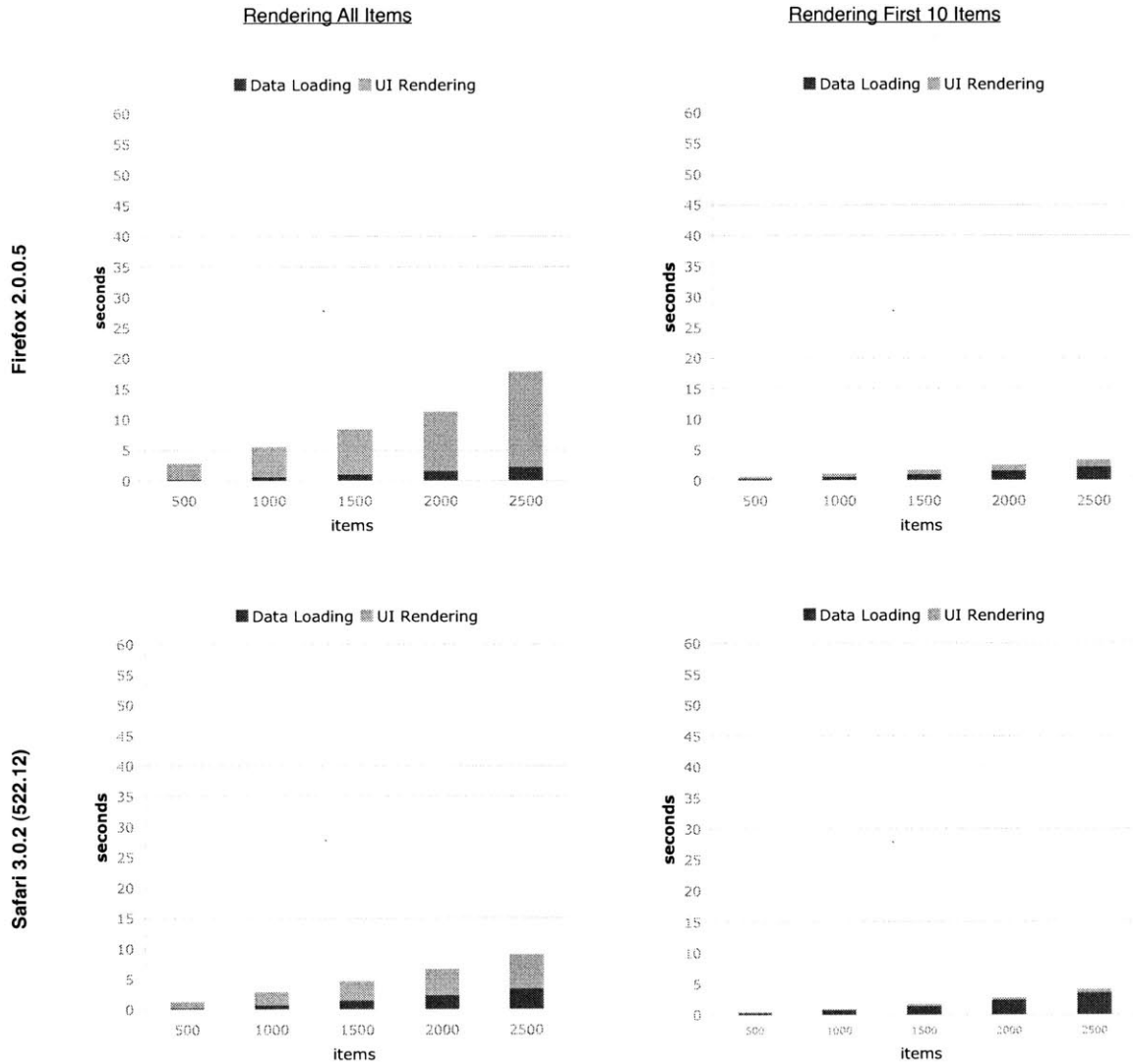
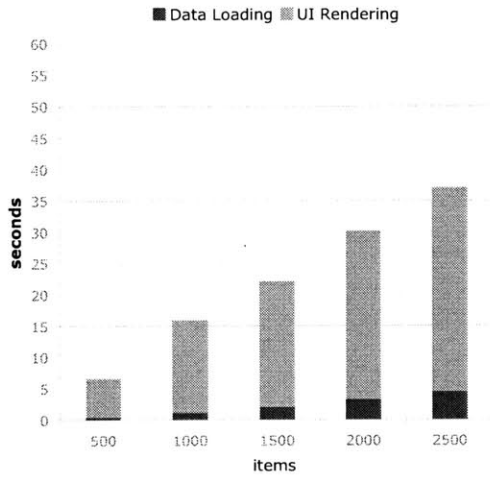


Figure 3.16. Exhibit's load-time performance results on four configurations (two operating systems, two browsers per operating system) with the number of items as the independent variable. At 1000 items, by rendering only the first 10 items, the data loading and UI rendering times combined take shorter than 5 seconds on all configurations.

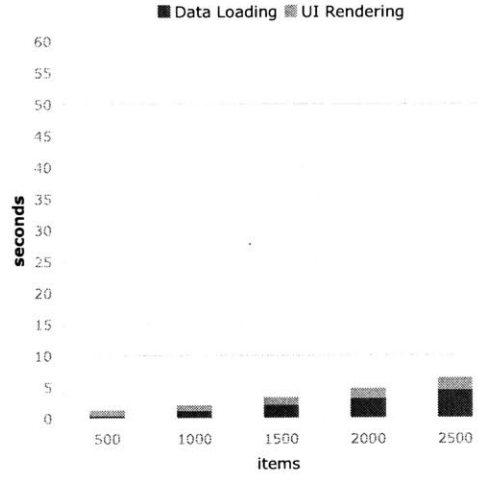
3. PUBLISHING DATA

Windows XP SP2, Pentium(R) 4 CPU 2.53GHz, 1.00 GB RAM

Rendering All Items

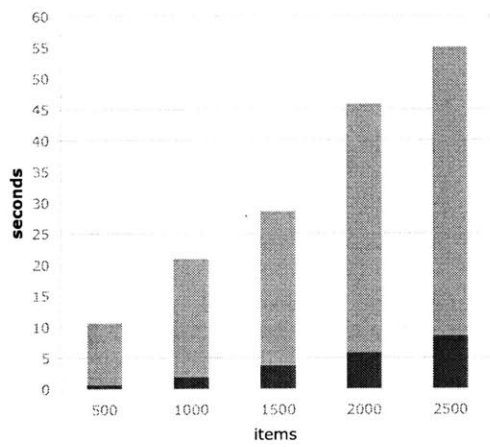


Rendering First 10 Items

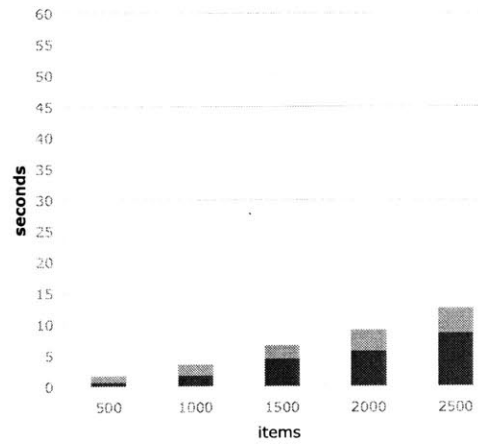


Firefox 2.0.0.5

Rendering All Items



Rendering First 10 Items



Internet Explorer 7.0.5730.11

3. PUBLISHING DATA

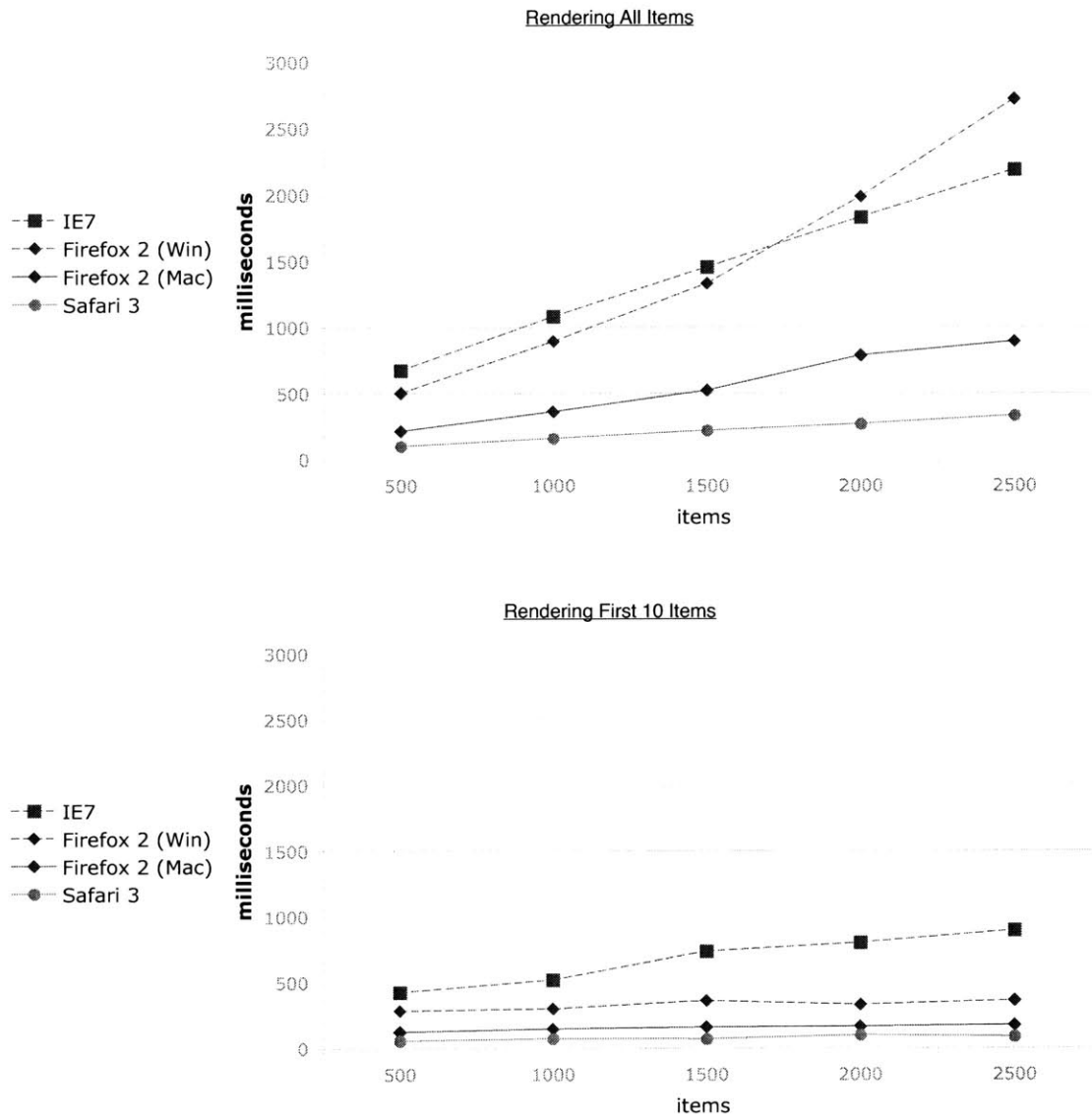


Figure 3.17. The times taken to filter down to 3 items from the same initial configurations in Figure 3.16. By rendering only the first 10 items initially, the filtering times on all configurations can be reduced to below one second even for exhibits containing 2500 items in total.

3.5.2 Usage

Eight months after Exhibit 1.0 was released, server logs indicate that more than 800 web pages link to the Exhibit API. Note that deploying Exhibit as a Web API serves two purposes: letting Exhibit authors do away with downloading and setting up software as well as allowing us to track actual usage of Exhibit through referrals in our server logs.

Table 3.1 shows usage of Exhibit at several top level domains. Many of them are attempts to follow a tutorial for creating exhibits. Some are broken and abandoned attempts; some are duplicate versions of the same exhibits; and a few are in foreign languages which I do not understand.

It is very difficult to categorize the rest as they are very unique and different from one another, as much as existing web sites are unique. An early attempt to group them by topics is shown in Table 3.2. Note the many esoteric topics that form the long tail distribution. Even within a single topic such as “people” there is diversity, ranging from university presidents, to army officers, to tavern keepers. These exhibits can contain as few as three items, to as many as 1,700. A few have been well designed visually, many have layouts and color themes copied off from other exhibits, and the rest are bare. Out of those 700 or so pages, 175 make use of the timeline view and 123 of the map view.

Figures 3.18 – 3.27 show ten exhibits created by real Exhibit authors. Figure 3.18 shows an exhibit that has been created by someone who claimed in an e-mail message to me not to be a programmer. Even though there is plenty of information online about music composers, he still felt compelled to create a “database” of music composers.

Figure 3.19 displays information about historical figures. It is meant to be classroom materials for grade six. Even though the same materials are available elsewhere, this teacher wanted to present them to his students in a different way, perhaps to match the rest of his own curriculum. Similarly, a Charmed TV series fan put up Figure 3.20 even an official episode guide exists.

On the other hand, the information shown in Figure 3.21—information about someone’s ancestors—will unlikely be published anywhere else, as it is very personal information. Similarly, the data about sweet products of a small Spanish shop shown in Figure 3.22 is unique.

Figure 3.23 shows an exhibit of banned books. By making it easier for end users to publish arbitrary data, Exhibit makes such information, shunned by official sources, more likely to see the light of day. From this perspective, Exhibit serves as a tool for empowerment.

Exhibit is also found to be a useful organization tool thanks to its faceted browsing and flexible visualization features: Figure 3.24 shows information about little league games and practices. Several tabular views are configured to show the same data differently, perhaps to different groups in the exhibit’s target audience. The four digit numbering scheme in the events’ labels indicates expected long-term use.

3. PUBLISHING DATA

More interestingly, Exhibit has been found useful even by programmers. The programmer responsible for implementing a browsing interface for the learning resources in the Language Learning and Resource Center at MIT chose Exhibit over building his own web application and have made several exhibits, including the one shown in Figure 3.25.

Similarly, a database researcher has found it easier to make an exhibit of his publications, seen in Figure 3.26, than to build his own database-backed publication site.

Finally, semantic web researchers have turned to Exhibit as a means to display their data (Figure 3.27), especially because they have no tool to display their data in such a usable and useful manner.

3.6 Summary

This chapter explored several ideas that made publishing data to the Web in browsable and reusable forms easy for casual users, those without programming skills. Embodying these ideas, the lightweight publishing framework Exhibit has let a few hundred people create richly interactive web sites with maps and timelines.

TLD	exhibit count
.com	246
.edu	215
.org	91
.net	35
.nl	20
.fr	19
.ca	18
.uk	17
.it	11
.info	7
.co	6
.ro	6
.de	5
.us	5
.mil	4
.at	3
.dk	3
.fi	2
.gov	2
.ws	2
.ar	1
.au	1
.cx	1
.cz	1
.gr	1
.hu	1
.in	1
.int	1
.is	1
.nu	1
.pl	1
.ru	1
.se	1
.za	1

Table 3.1. A survey of real-world exhibits by topics shows adoption in many top level domains.

topic	exhibit count	item count
publications	20	4013
people	18	2647
scientific data	6	2517
photos	5	871
classroom materials	5	477
engineering timetables	4	899
products	3	433
movies	3	237
sports	3	195
events	3	162
presentations	3	128
projects	2	957
web sites	2	317
books	2	199
locations	2	127
recipes	2	97
personal histories	2	12
play castings	1	441
games	1	422
space launch sites	1	280
world conflicts	1	278
hotels	1	203
restaurants	1	97
blog posts	1	97
cars	1	89
links	1	70
buildings	1	68
breweries & distilleries	1	55
elections	1	50
activities	1	50
museums	1	45
sounds	1	32
historical artifacts	1	24
characters	1	24
streets	1	24
early christian periods	1	21
HMong state population	1	15
vocabulary	1	14
history	1	13
materials	1	8
documents	1	7
plays	1	5
wars	1	3

Table 3.2. A survey of real-world exhibits by topics shows a long tail distribution.

3. PUBLISHING DATA

The screenshot shows a web browser window with the URL <http://www.musicedmagic.com/Exhibit/composers.html>. The page features a large header with a treble clef icon and the text "Music Education Magic" and "Music and General Education News, Information, and Resources For Instrumental Music Students and Their Teachers". Below this is the title "Music Composer Research Database".

The main content area includes a paragraph: "The information in this database is intended to be a starting point for research and study into the great composers of classical music." and another paragraph: "The information about each composer is provided in a clean, visual format that will help you see the relationships between the various composers of the Medieval Period, the Renaissance, the Baroque Era, the Classical Era, the Romantic Era, and current Twentieth Century music. For more detailed information on how to use this database please visit the [instruction manual at MusicEdMagic.com](#)".

Navigation links include "Ads by Google", "Devon Family History", "Somerset Genealogy", "Hoff Genealogy", and "Hough Genealogy". A "VIEW TIMELINE" link is highlighted, along with "VIEW ALL INFO", "MAP PLACES OF BIRTH", "PLACE OF DEATH", and "PLACE OF BURIAL".

A "64 Items" section shows a timeline visualization with horizontal bars representing the lifetimes of composers. Visible names include John Townner Williams, Phillip Glass, John Cage, Leonard Bernstein, Frank Tichelli, George Gershwin, and Michael Kamen. A note states "Only 55 can be plotted on the timeline." A "Copy All" button is present.

On the right side, there are three filter menus: "Era" (Baroque, Classical, Middle Ages, Renaissance, Romantic, Twentieth Century, Uncategorized), "Major Genres" (Aleatoric, American Folk, Arias, Ballet, Blackface Minstrelsy, Carons, Carteras), and "BirthCountry" (Australia, Austria, Belgium).

Figure 3.18. An exhibit of 64 music composers from the eleventh century to the present day. The composers can be viewed on a timeline by their lifetimes, and on maps by their birth places, death places, and burial places. They can be browsed by the music eras, by their major genres, and by their countries of birth. Even with all the information about composers already on the Web, someone still feels compelled to create his own research “database” of composers.





SIMILE | Exhibit | 6th Grade SOLs
 http://byrdmiddle.org/sol.html

Important 6th Grade SOL Figures

This is more than a web page. This is an interactive site that allows you to manipulate and sort the data with just a click or two. If you'd like to see an example of the page in use [click here](#).

TABLE • DETAILS • PHOTOS • BIRTH PLACES • SITES OF DEMISE • LIVES Copy All

19 People total

NAME	PICTURE	ROLE	ERA	BIRTH	DEATH	FACTS	SOL
John Cabot		Explorer	Exploration	1450-01-01	1499-05-22	John Cabot explored eastern Canada.	USI.4a
Francisco Coronado		Explorer	Exploration	1510-01-01	1554-01-01	Francisco Coronado claimed the southwest United States for Spain	USI.4a
Samuel de Champlain		Explorer	Exploration	1567-01-01	1635-12-25	Champlain established the French settlement of Quebec.	USI.4a
John Locke		Philosopher	Revolutionary War	1632-07-29	1704-10-28	Locke was an English philosopher who believed that people had inherent rights to life, liberty and property. His beliefs that the government	USI.6b

ROLE

- 4. Explorer
- 1. Journalist
- 1. Patriot
- 1. Philosopher
- 1. Poet
- 2. Politician
- 6. President

ERA

- 2. Civil War
- 5. Colonial
- 4. Exploration
- 2. New Nation
- 6. Revolutionary War

Done

Figure 3.19. An exhibit of 19 important figures in history that serves as educational materials in a grade 6 classroom. Even though the same materials are available elsewhere, each teacher might want to present them to their students in a different way, adding personal insights that match the rest of their curriculum.

3. PUBLISHING DATA

The screenshot shows a web browser window titled "Charmed TV Series Episode Guide" with the URL "http://www.storycharms.com/tvseries.html". The page features a navigation menu on the left with links like "Charmed Episode Guide", "Charmed Timeline", and "Charmed Series Auction items". The main content area is titled "Charmed TV Series Episode Guide" and contains a list of 178 items. The first six episodes are visible, each with a title, season, episode number, date, and a brief synopsis. A small image of the three main characters is shown next to the first episode. On the right side, there are two vertical scrollable lists: "features" and "episode". The "features" list includes items like "Agent Brody", "Alchemist", "Andras", "Andy", "Angel of Death", "Angel of Destiny", and "Angel of Fortune". The "episode" list includes "1.01", "1.02", "1.03", "1.04", and "1.05". A "Rate this page" section is located in the bottom left of the main content area, with radio buttons for "Excellent", "Good", "Fair", and "Poor".

Charmed TV Series Page

Charmed TV Series Episode Guide

See episode details of Charmed, your favourite TV series, right here. Click on the options to change the information being shown on the page. There's a lot of information on this page, so please allow time for the details to load.

178 Items total Copy All

sorted by: episode; then by... grouped as sorted show duplicates

- Something Wicca this Way Comes** | Season: 1 | Episode: 1.01 | Date: 01/04/2000
The Halliwell sisters are reunited and discover their heritage much to their dismay.
- I've Got You Under My Skin** | Season: 1 | Episode: 1.02 | Date: 08/04/2000
Piper is scared to go into a church now she is a witch. Phoebe meets a photographer but he's really a youth-sucking demon after immortality.
- Thanks for Not Morphing** | Season: 1 | Episode: 1.03 | Date: 15/04/2000
The sisters' absent father appears wearing an anti-evil amulet. And their neighbours are trying to steal the Book of Shadows.
- Dream Sorcerer** | Season: 1 | Episode: 1.04 | Date: 22/04/2000
Prue has terrifying dreams of a man threatening to kill her and he nearly does.
- Dead Man Dating** | Season: 1 | Episode: 1.05 | Date: 29/04/2000
A ghost asks Piper to ensure his body is buried to save him from a Chinese demon.
- The Wedding from Hell** | Season: 1 | Episode: 1.06 | Date: 06/05/2000
Piper is catering a wedding but then the bride is suddenly jilted for a stranger.

features

- 12 ✓ Agent Brody
- 1 ✓ Alchemist
- 1 ✓ Andras
- 2 ✓ Andy
- 2 ✓ Angel of Death
- 1 ✓ Angel of Destiny
- 1 ✓ Angel of Fortune

season

- 22 ✓ 1
- 22 ✓ 2
- 22 ✓ 3
- 22 ✓ 4
- 23 ✓ 5
- 23 ✓ 6
- 22 ✓ 7

episode

- 1 ✓ 1.01
- 1 ✓ 1.02
- 1 ✓ 1.03
- 1 ✓ 1.04
- 1 ✓ 1.05

Rate this page

- Excellent
- Good
- Fair
- Poor

Done 1 Error

Figure 3.20. An exhibit of 178 episodes from the Charmed TV series.

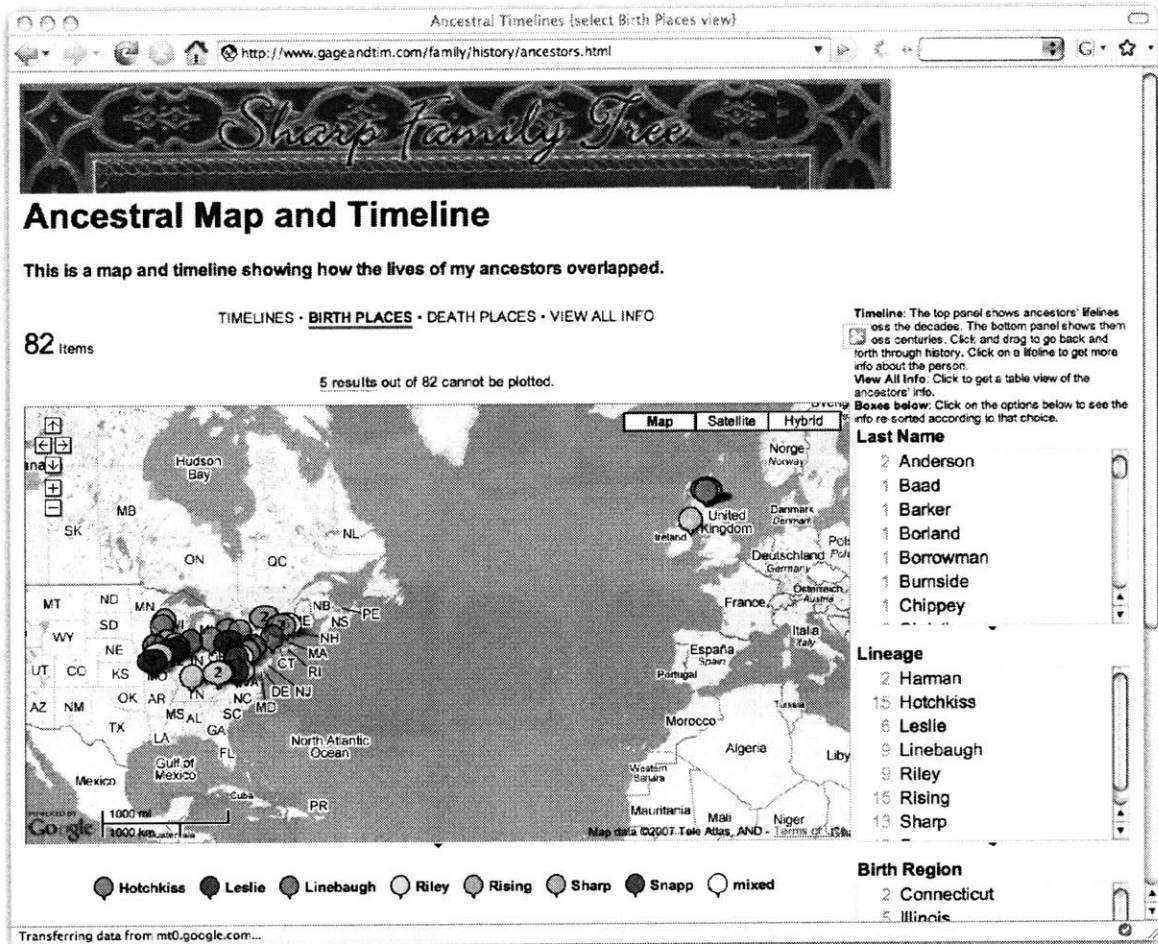


Figure 3.21. An exhibit of 82 ancestors containing a timeline and maps of birth places and death places. The people can be browsed by their last names, lineage, birth regions, and death regions. The information found in this exhibit will unlikely be on any other web site. That is, even with Wikipedia and Freebase growing in size everyday, some information will just never find its way into such public repositories.

3. PUBLISHING DATA

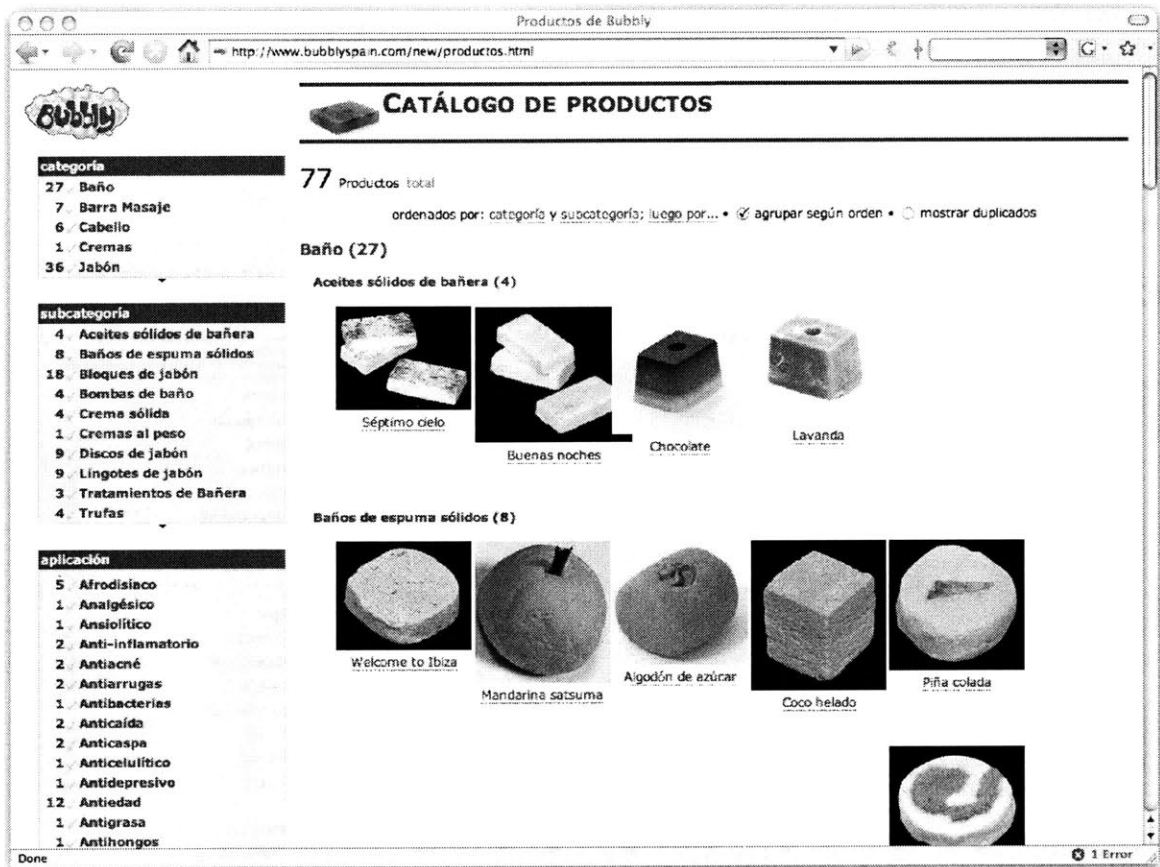


Figure 3.22. An exhibit of 77 sweet products in a Spanish shop. For such a small business, investing in a full three-tier web application is costly.

Syracuse University Library :: Banned Book Week

http://library.syr.edu/information/banned_books/exhibit.html

SYRACUSE UNIVERSITY LIBRARY

ABOUT US SERVICES HELP RESEARCH TOOLS

Celebrate Your Freedom - Read a Banned Book

Syracuse University Library observes Banned Books Week, September 29 - October 6, 2007, as a part of the S.J. Newhouse School of Public Communications year-long First Amendment Celebration. For more information about Banned Books Week, contact Tasha Cooper, nacoop01@syr.edu or visit the resource pages hosted by the American Library Association.

11 items total

[THUMBNAILS](#) • [DETAILS](#) • [TIMELINE](#)

sorted by: type; then by... • grouped as sorted • show duplicates

Adult Book (3)

<p>Eve's Diary by Mark Twain and Lester Ralph Banned/Challenged: Illustrations</p>	<p>Lolita by Vladimir Nabokov Banned/Challenged: Sexual Situations</p>	<p>The Bluest Eye by Toni Morrison Banned/Challenged: Sexual Situations, Strong Language, and Racism</p>
---	---	---

Children's Book (4)

<p>Lango book 3</p>	<p>A Wrinkle in Time</p>	<p>American Girl</p>
--------------------------------	---------------------------------	-----------------------------

type

- Adult Book
- Children's Book
- Essay
- Pamphlet
- Play

reason

- Illustrations
- Political Views
- Racism
- Religious Views
- Sexual Situations
- Strong Language

year

- 1829
- 1848
- 1849
- 1906
- 1955
- 1962
- 1970

location

- Bird Library
- Bird Library Special Collections
- MLK Library

Dcne

Figure 3.23. An exhibit of 11 banned books. Certain materials may never find their way into official discourse. Empowering end users to publish is even more important a cause.

3. PUBLISHING DATA

Cheltenham Little League Events (select Practice and Game Table view)

http://www.cheltenhamlittleleague.

Cheltenham Little League Events

+ Instructions Select your team in "All_Team_Events" below to get all of the events for your team!

Date: 2/01/02/07, 2/01/03/07, 1/01/05/07, 2/01/09/07, 1/01/10/07, 1/01/22/07, 1/02/01/07

Sport: 3/ baseball, 5/ softball, 2/ t-ball

Event: 3/ game, 6/ practice, 1/ tournament

All_Team_Events: 3/ CLL12, 1/ CLL20, 2/ CLL5, 3/ CLL6, 1/ EA12, 1/ EA4, 1/ JYA4

PRACTICE AND GAME TABLE • UMPIRES TABLE • GAME, PRACTICE AND UMPIRE TILE

10 Items total Copy All

Label	Date	Start	Sport	Event	Location	Field	Home	Away	Practice	All Teams
0001	01/02/07	09:00	baseball	practice	CAA	CAA1			CLL12	CLL12
0002	01/02/07	10:30	baseball	practice	CAA	CAA3			CLL5	CLL5
0003	01/03/07	10:00	softball	game	OYRLL	OYR1	CLL12	EA4		EA4 and CLL12
0004	01/03/07	13:30	t-ball	practice	OYRLL	OYR1			CLL6	CLL6
0005	01/05/07	14:00	softball	game	EA	EA3	EA12	CLL12		CLL12 and EA12
0006	01/09/07	10:00	t-ball	practice	OYRLL	OYR2			OPEN	OPEN
0007	01/09/07	11:00	softball	game	CAA	CAA2	CLL5	CLL6		CLL6 and CLL5
0008	01/10/07	15:30	baseball	practice	CAA	CAA2			OPEN	OPEN
0009	01/22/07	14:00	softball	tournament	CAA	CAA1	CLL20	JYA4		JYA4 and CLL20
0010	02/01/07	14:00	softball	practice	OYRLL	OYR4			CLL6	CLL6

Done

Figure 3.24. An exhibit for organizing little league sport games. It is filled with acronyms and is tailored for a very small audience, to whom it proves useful.

LLARC Catalogue

http://llarc.mit.edu/materials/catalogue/

Location: LLARC > Materials > LLARC Catalogue > LLARC Catalogue

LLARC MIT Language Learning and Resource Center

Materials in the LLARC Catalogue must be checked out and used in 16-644. For materials you can check out to use at home, see the Grandall Collection.

Entire Catalogue

CBES Chinese ESL French German Japanese Spanish

SEARCH:

AUTHOR OR DIRECTOR	MEDIA TYPE	LANGUAGE	AREAS
109	23 Audio CD	1	69
1 ABC News	224 Cassette	23 Bilingual/Bicultural Studies	64 Culture
1 Aciman, André	28 CD-ROM	87 Chinese	374 Films
1 Adams, Thomas and Kuder, Susan	5 Computer Program	118 ESL	1 Films ; Culture
4 Akin, Fatih	198 DVD	144 French	20 Films; Culture
1 Alarcon, Ruiz de	4 Text only	228 German	1 Films; CultureFilmsCulture
1 Alassane, Mustapha	6 Video CD	1 Italian	

LLARC CATALOGUE • SHOW LONG DESCRIPTIONS

867 Items

CAT. NO.	TITLE	AREAS	AUTHOR/DIRECTOR	MEDIA	COPIES
CH 077	2046	Films	Wong, Kar Wai	DVD	1
FR 175	A BATONS ROMPUS	Language Study; Lectures, Speeches, Interviews	Furstenberg, Gilberte	Cassette	

Figure 3.25. An exhibit of 867 teaching resources in the Language Learning and Resource Center at MIT. Its author is skilled in server-side programming but he still finds Exhibit a quick way to build a browsable interface for his data.

3. PUBLISHING DATA

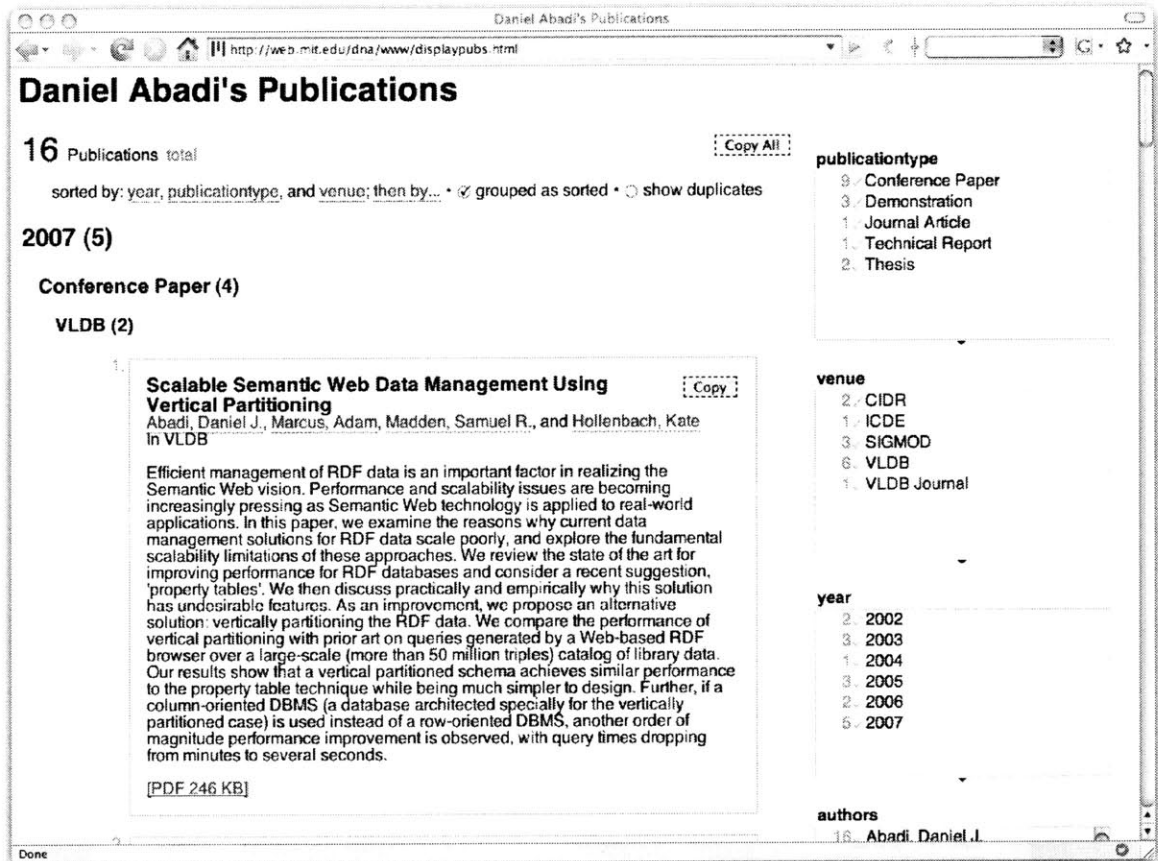


Figure 3.26. An exhibit of a database researcher's publications, who finds it easier to use Exhibit than to build a database-backed web site for his publications.

Semantic Web Education and Outreach Interest Group Case Studies and Use Cases

http://www.w3.org/2001/sw/sweo/public/UseCases/Overview.html

W3C Technology and Society domain Semantic Web Activity

Semantic Web Education and Outreach Interest Group: Case Studies and Use Cases

Case studies include descriptions of systems that have been deployed within an organization, and are now being used within a production environment. Use cases include examples where an organization has built a prototype system, but it is not currently being used by business functions.

The list is updated regularly, as new entries are submitted to the Interest Group. There is also an RSS1.0 feed that you can use to keep track of new submissions.

22 entry

sorted by: entry-type and labels; then by... grouped as sorted

Case study (13)

1. [An Intelligent Search Engine for Online Services for Public Administrations, Municipality of Zaragoza \(Case study\)](#)
Contributed by: Jesús Fernández Ruiz
2. [An Ontology of Cantabria's Cultural Heritage, Fundación Marcelino Botín \(Case study\)](#)
Contributed by: Francisca Hernández
3. [Composing Safer Drug Regimens for the Individual Patient using Semantic Web Technologies, PharmaSURVEYOR Inc. \(Case study\)](#)
Contributed by: Erick Von Schweber
4. [Enhancing Content Search Using the Semantic Web, Siderean Software and Oracle Corporation \(Case study\)](#)
Contributed by: Mike DiLascio and Justin Kestelyn
5. [Geographic Referencing Framework, Ordnance Survey \(Case study\)](#)
Contributed by: Catherine Dolbear
6. [Improving the Reliability of Internet Search Results Using Search Thresher, Segala \(Case study\)](#)
Contributed by: David Rooks
7. [Real Time Suggestion of Related Ideas in the Financial Industry, Bankinter \(Case study\)](#)
Contributed by: José Luis Bas Unbe
8. [Semantic Content Description to improve discovery, Vodafone Group Research & Development \(Case study\)](#)
Contributed by: Kevin Smith
9. [Semantic Web Technology for Public Health Situation Awareness, School of Health Information Sciences, University of Texas \(Case study\)](#)
Contributed by: Raza Mirbani

Search facets:

Application

- 2 B2B Integration
- 8 business organization
- 1 cultural heritage
- 8 data integration
- 1 eGovernment
- 1 geographic information system

Country

- 1 Belgium
- 1 China
- 1 France
- 1 India
- 1 Ireland
- 1 Italy
- 6 Spain

Institution's Activity area

- 1 aeronautics
- 1 automotive
- 1 financial institution
- 1 health care

Figure 3.27. An exhibit of Semantic Web case studies and use cases. Exhibit is making Semantic Web data useful by making it viewable by end users.

3. PUBLISHING DATA

4. EXTRACTING DATA

Just as Exhibit has wrapped up much complexity in the conventional three-tier web application architecture in order to making data publishing easy enough for casual users, to bring web data extraction technologies to casual users we must also wrap them up in a ready-to-use package.

- First, some set of commonly needed features such as sorting and filtering can be provided out-of-the-box.
- Second, to keep the user’s visual context from before to after extraction, and to take advantage of already well designed visual elements from the original web page, the presentation elements in the original page are reused as much as possible. In particular, when a data record is extracted from a part of the original page, that fragment of the page is also extracted so that in order to show that data record later on, we can simply show the fragment again. The rest of the web page, which does not contain data to extract, is kept as-is.
- Finally, direct manipulation “handles” can be inserted right into the web page, next to each data value, so that the user can essentially grab a data value and interact with it, such as invoking a sorting or filtering command. This application of direct manipulation lets us do away with extracting field labels or with requiring the user to label the fields herself.

These ideas have been built into a browser extension called Sifter that can augment a sequence of web pages (typically search result pages) *in-place* with filtering and sorting functionality while requiring from the user as few as two clicks. The remainder of this chapter first describes the user interface of Sifter from the extraction phase to the augmentation phase. Next, the data extraction algorithm is explained. Finally, evaluations of the algorithm and the user interface are reported and discussed.

4.1 User Interface Design

User interaction with Sifter consists of two stages:

- extraction, when the system ascertains which parts of the web site to extract, and gives the user feedback about this process; and
- augmentation, when the system adds new controls to the web page and the browser that allow the user to filter and sort the extracted items in-place.

4.1.1 Extraction User Interface

Sifter's user interface resides within a pane docked to the right side of the web browser (Figure 4.1). When the user first visits a web site, the Sifter pane shows a single button that, when clicked, triggers the detection of items on the current web page as well as links to subsequent pages, if any. An item is, for example, a product as in Figure 4.1. Items are highlighted in-place, and the total number of items spanning the detected series of pages is displayed prominently in the pane. If the system has incorrectly detected the items or the subsequent-page links, the user can correct it by clicking on an example item or a subsequent-page link in the web page. Once the **Continue** button (Figure 4.1) is clicked, the extraction process starts and Sifter pops up a dialog box showing the subsequent web pages being downloaded. Over all, getting the data extracted usually takes two button clicks.

During the extraction process, Sifter locates all items on all the web pages, extracts field values from each item as well as its HTML code, and stores each item as a record in a local database. For example, the local database accumulated from extracting the 7 pages of 59 items in Figure 4.1 would contain 59 records, each having one text field (title), two numeric fields (price and percent saving), and an HTML code fragment representing the entire item. This code fragment is used as a rendering of the item when the result set is filtered or sorted.

4.1.1.1 Making Corrections

If the items are detected incorrectly, the user can click on the **Locate Items** button (Figure 4.1) and the Sifter pane will change to highlighting mode (Figure 4.2). In this mode, as the user moves the mouse cursor over the web page, the system inspects the smallest HTML element under the mouse cursor, generalizes it to other similar elements on the page, expands those elements to form whole items, and highlights these candidate items with translucent overlays. When the user is satisfied with the highlighting, she can click the mouse button and the Sifter pane switches out of its highlighting mode.

There was consideration to eliminate the extraction UI altogether and provide a correction UI after the extraction process has finished. However, as the extraction process may be lengthy and not completely reliable, providing a preview of what the system is going to do makes the wait more acceptable and gives the user a sense of greater control over the system.

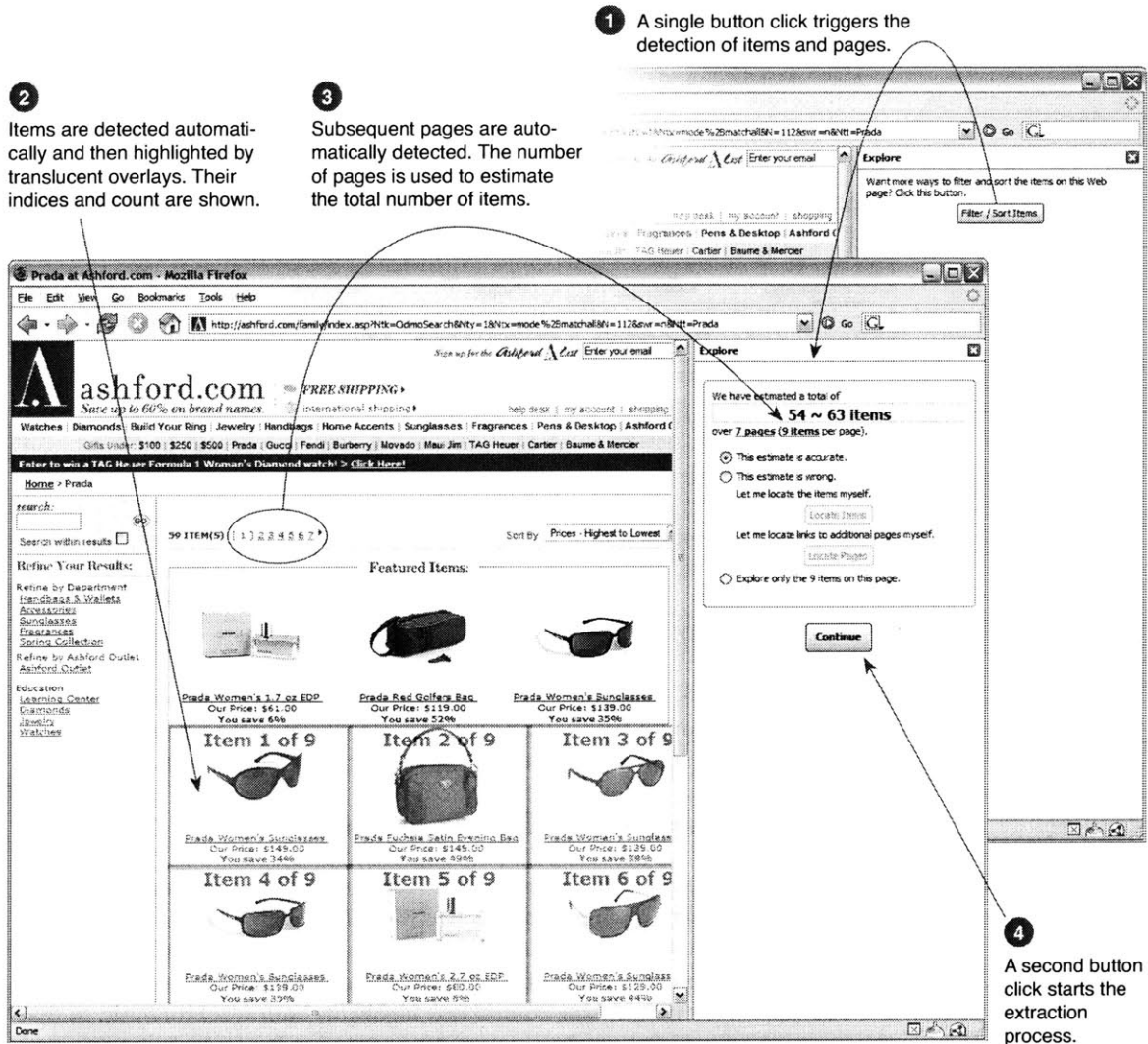


Figure 4.1. Sifter provides a one-click interface for triggering the detection of items and pages. One more button click starts the extraction process.

4. EXTRACTING DATA

Candidate items are highlighted and their indices and count are shown as watermarks.

Position of mouse pointer helps identify items to be extracted.

The whole Web page is shown in miniature for an overview of the highlighted items, bringing attention to misses evident as whitespace gaps, if any.

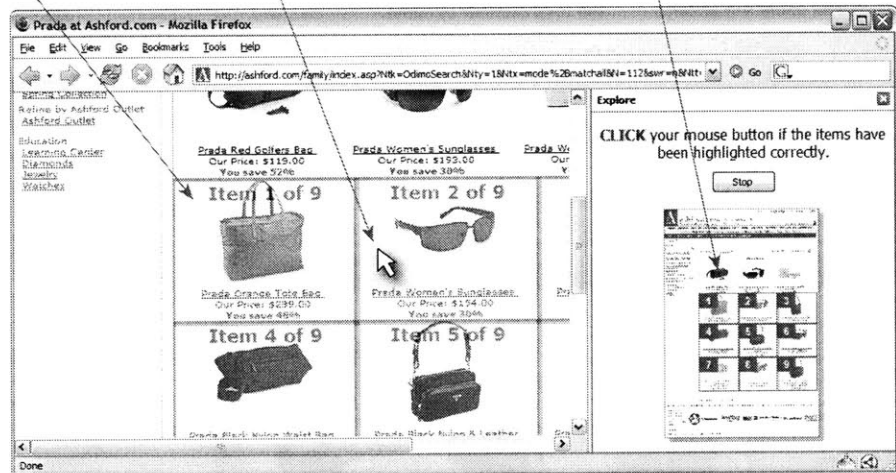


Figure 4.2. Sifter's highlighting mode provides an interactive mechanism for the user to correct the automatic detection of items by hovering the mouse pointer over the web page and clicking once the items have been highlighted correctly.

4.1.2 Augmentation User Interface

Figure 4.4 shows Sifter in action as the user makes use of new filtering and sorting functionality. An asterisk is inserted after each field value in the web page. When an asterisk is clicked, a *browsing control box* is displayed in the Sifter pane, containing the filtering and sorting controls for that field. Hovering the mouse pointer over either a field's asterisks or its browsing control box highlights both synchronously, so that the user can tell which browsing control box corresponds to which asterisk, and hence, which field. By keeping the original presentations of the items, Sifter can provide affordance for applying direct manipulation techniques on the field values without ever knowing the field labels.

As the user invokes filtering and sorting commands, Sifter dynamically rewires the web page to show the set of items satisfying the current filters in the current sorting order, as if the web site itself had performed the filtering and sorting operations. Sifter does so by removing the HTML fragment of each item on the page and then injecting into the same slots (where those removed fragments previously fit) the HTML fragments of the items satisfying the current dynamic query. These HTML fragments are retrieved from the local database, so there is no need to make a request to the original site when the dynamic query changes.

Items satisfying the current dynamic query are inserted into the original Web page as if the Web site itself has performed the query.

Extraneous content (e.g., sponsor links) is faded away to avoid confusion and distraction.

Sorting controls for a field

Paging controls for the whole collection.

The screenshot shows a Mozilla Firefox browser window displaying an Amazon.com search results page for 'Jeffrey Archer'. The page is annotated with several callouts. One callout points to the search results, stating that items satisfying the current dynamic query are inserted into the original web page as if the website itself has performed the query. Another callout points to a faded sponsor link, stating that extraneous content is faded away to avoid confusion and distraction. A third callout points to the sorting controls for a field, and a fourth callout points to the paging controls for the whole collection. A 'Sifter' pane is open on the right side of the page, showing filtering options for 'Audio Cassette', 'DVD', 'Hardcover', and 'Paperback', and a date range filter. The Sifter pane also shows a table of items filtered by date range.

Value	#items
Audio Cassette	1
DVD	0
Hardcover	8
Paperback	19

Value	#items
2000 to 2010	11
2001	1
2003	3
2004	2
2005	4
March 2005	1
July 2005	2
November 2006	1

An asterisk is inserted after each field value. Clicking on an asterisk adds a browsing control box to the Sifter pane. Corresponding asterisks and boxes are co-highlighted when hovered.

Figure 4.4. After extraction is complete, the Sifter pane hosts filtering and sorting controls, which when invoked, re-render the resulting items inside the same web page (without needing to contact the original web server again).

Since exploring the collection of items may involve clicking a link to view details about an item, Sifter stores the query state and automatically restores it when the user returns to the augmented page.

4.1.2.1 Filtering

The filtering controls for different field types (text, numbers, date/time) manage the field values differently. For numbers and date/time fields the values are classified into ranges and sub-ranges hierarchically, while for text fields the values are listed individually. Selecting a value or a range filters the current collection of items

4. EXTRACTING DATA

down to only those having that value or having values in that range. Multi-selection in a single field adds disjunctive query terms. Filtering on more than one field forms conjunctive queries. Selecting and de-selecting field values or ranges in a browsing control box updates the available values and ranges in other boxes as in any dynamic query interface [67].

Note that filtering in Sifter is equivalent to faceted browsing in Exhibit (the different terminologies are an artifact of the evolution of my research). Whereas in Exhibit each facet has a label (taken from the corresponding property's label or assigned explicitly by the publisher), in Sifter none of the browsing control boxes has label. Furthermore, in Exhibit, a facet's class (list or numeric range) and placement are determined by the publisher who knows the data and is interested in making her exhibit look and behave however she feels appropriate. In contrast, a Sifter user does not know a web page's data as well and is only interested in sorting or filtering the items therein. For that reason, Sifter takes a more utility-focused than creativity-focused approach: the user is not given control over how the browsing control boxes look and their classes (list, numeric range, or date range) are chosen automatically as much as possible.

4.1.2.2 *Visual Context*

While the user is using Sifter's filtering and sorting functionality, the rest of the original web page is kept for three reasons:

- To maintain visual context so that the user does not have to orient herself in a completely new view of the same information she was seeing before invoking Sifter.
- To avoid throwing away vital information that helps the user understand the items. For instance, the fact that the sale prices are only applicable to students might not be embedded in every item's presentation but is isolated in another part of the page. It would be hard to extract that piece of information.
- To make features offered by the original page still accessible to the user.

However, while preserved, the rest of the page is faded out to visually indicate a disconnection between the features in the original page and the items under Sifter's control. The original status indicators (e.g., number of items, number of pages) are faded to imply that they no longer apply to the items inside the web page. The original pagination, sorting, and browsing controls are faded to imply that invoking them would switch out of Sifter's augmentation mode and let the user interact with the original web site.

4.2 Data Extraction

Given the user interface design described above that requires minimal user intervention during the extraction phase, this section explains how the data extraction algorithm is designed to support that interface. The extraction algorithm consists of three steps: locating items to extract; identifying subsequent web pages; and parsing useful field values from each item.

4.2.1 Item Detection

Let us posit that for many sequences of web pages containing lists of items (e.g., search results, product listings), there exists an XPath [38] that can precisely address the set of items on each page. Thus, our item detection algorithm involves deriving such an XPath. (The cases where each item consists of sibling or cousin nodes [78] will be addressed in future work.) Let us also posit that this XPath can be computed just from the sample items on the first page in a sequence of pages. These assumptions are expected to generally hold on database-backed web sites that generate HTML from templates as opposed to hand-writing their HTML.

The algorithm is based on two observations. First, in most item collections, each item contains a link, often to a detail page about the item. So, links are likely to be useful as starting points for generating hypotheses for the XPath. Second, the item collection is typically the main purpose of the web page, so the items themselves consume a large fraction of the page's visual real-estate. This gives us a way to choose the most likely hypothesis, namely, the one that uses the largest area of the page.

The item detection algorithm starts by collecting all unique XPaths to **<A>** elements on the current web page. For each element, its XPath is calculated by stringing together the tag names of all elements from the document root down to that element. CSS class names are also included. The resulting XPath looks something like this: `/HTML/BODY/TABLE/TBODY/TD/DIV[@class='product']/SPAN/A`. Each such XPath is general enough to cover more than just the original **<A>** element, but restrictive enough to address only those elements similar to it. This is called the generalization phase, as illustrated by the straight arrows in Figure 4.3.

Each of these **<A>** XPaths addresses a collection of **<A>** elements that could correspond one-to-one with the collection of items to be detected. We wish to find which of these **<A>** XPaths corresponds to a collection of items that take the largest amount of screen space.

Next, each **<A>** XPath is expanded to fully encompass the hypothetical items that the **<A>** elements reside within. To expand an XPath, we repeatedly append `/. .` to it (see the curved arrows in Figure 4.3). As `/. .` is appended, the set of HTML elements that the XPath addresses gets closer and closer to the document root. As long as the cardinality of that set remains unchanged, each HTML element in that set still resides spatially inside a hypothetical item. When the cardinality of the set drops, the XPath has been expanded too much such that it now describes the par-

4. EXTRACTING DATA

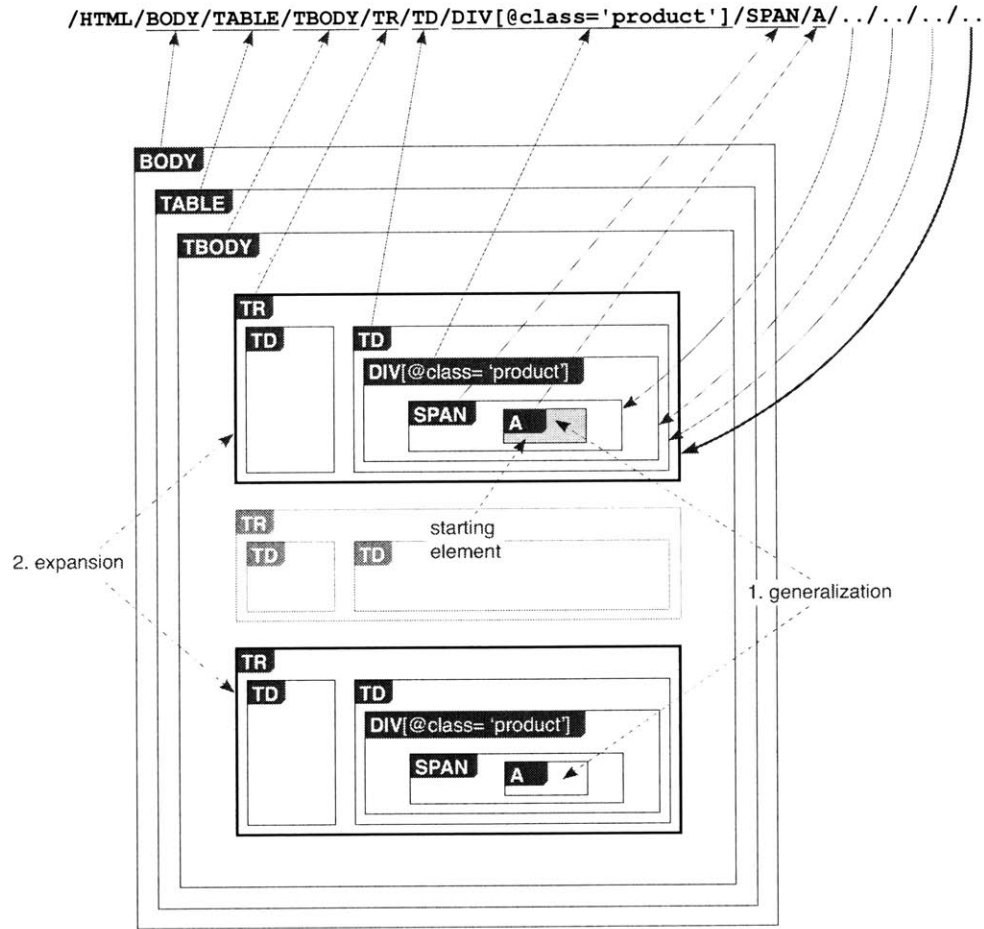


Figure 4.3. Given a starting HTML element, an item XPath is constructed by generalization to similar elements (straight arrows) and then expansion (curved arrows).

ent node(s) of the hypothetical items. For example, in Figure 4.3, if we append another `/..`, the resulting XPath would address a single `TBODY` element rather than two `TR` elements. We stop appending `/..` just before that happens. The result is a candidate item XPath.

Note that we append `/..` rather than truncate ending segments because truncation loses information. If the XPath in Figure 4.3 were instead truncated 4 times, the resulting XPath, `/HTML/BODY/TABLE/TBODY/TR`, would have included the middle `TR`, which does not have a link inside and could be extraneous, intervening content. Using `/..` guarantees matching only nodes that have the link inside them.

For each candidate item XPath, we calculate the total screen space covered by the HTML elements it addresses. The candidate item XPath with the largest screen space wins and is then used to add highlight overlays to the web page.

4.2.2 Subsequent-Page Detection

Two heuristics are used to automatically detect subsequent pages. The heuristics are run in the order presented below, and when one succeeds, its results are taken as final.

4.2.2.1 Link Label Heuristic

Often, a web page that belongs in a sequence of pages contains links to the other pages presented as a sequence of page numbers, e.g.,

Pages: 1 [2] [3] [4] [5] [Next] [Last]

Occasionally, such a sequence shows not the page numbers but the indices of the items starting on those pages, e.g.,

Items: 1–10 [11–20] [21–30]

This heuristic attempts to pick out URLs from such a sequence of linearly increasing numbers. First, the text labels of all **<a>** links on the current web page are parsed. Only labels that contain numbers are kept. They are then grouped by the XPaths generated from the **<a>** elements. For each XPath, the numbers parsed from the labels are sorted in ascending order. Only those XPaths with linearly increasing sequences of numbers are kept. These final candidates are then sorted by the lengths of their sequences. The XPath with the longest sequence is then used to pick out URLs to subsequent pages. If there is a tie, the XPath whose sequence increases at the highest rate wins. This heuristic fails if no XPath has a linearly increasing sequence of numbers.

4.2.2.2 URL Parameter Heuristic

URLs of pages in a sequence often encode the page numbers or the starting item indices as numeric URL parameters. For instance, Amazon.com encodes page numbers in the **page** parameter and Yahoo.com encodes starting item indices in the **b** parameter (Table 4.1). This heuristic attempts to detect such parameters so that URLs to subsequent pages can be generated. The URLs pointed to by all the links on the current web page are parsed to extract out URL parameters. For each parameter that has numeric values, its numeric values are collected in an array and sorted in ascending order. Then, only parameters whose values form linearly increasing sequences are kept. These final candidates are sorted by the lengths of their value sequences. The parameter with the longest sequence is then used to generate URLs to subsequent pages. If there is a tie, the parameter whose sequence increases at the highest rate wins. This heuristic fails if no parameter has a linearly increasing sequence of values.

If these heuristics fail then the user can intervene and point at the link to one of the subsequent pages (not necessarily the immediately following page). The XPath of that link is then computed, which describes a collection of **<a>** elements. Given such a collection of **<a>** elements, the following heuristic is used to pick out the one that points to the next page.

4. EXTRACTING DATA

4.2.2.3 Next Page Heuristic

Table 4.1 gives some sample URL spaces of contemporary web sites. Pages in a sequence might not differ by only one URL parameter which encodes either the page number or the index of the starting item. In some cases, more parameters are inserted (e.g., `%5Fencoding=UTF8` at Amazon.com) and some existing ones are removed (e.g., `search-alias=aps` at Amazon.com). In other cases, URL parameters are not used at all (e.g., at Dogpile.com). Rather, some segments of the URL are used to specify the current page. Worse yet, the whole URL of the current page is encoded as a single URL parameter to another domain for some tracking purpose (e.g., at Yahoo.com).

The next page heuristic sorts candidate URLs together with the current page’s URL and picks out the URL immediately “larger” than the current page’s URL. Simple string sorting does not work as `page=10` will be “less” than `page=9`. Instead, each URL is broken into fragments separated by `/`, `?`, and `&`. Then the URLs are sorted by comparing corresponding fragments that contain numbers. (This heuristic cannot handle Yahoo’s URL space as shown in Table 4.1.)

Amazon.com (search for “metallica”)	
Page 1	<code>http://www.amazon.com/gp/search/ref=br_ss_hs/103-8791841-3931026?search-alias=aps&keywords=metallica</code>
Page 2	<code>http://www.amazon.com/gp/search/103-8791841-3931026?%5Fencoding=UTF8&keywords=metallica&rh=i%3Aaps%2Ck%3Ametallica&page=2</code>
Dogpile.com (search for “haystack”)	
Page 1	<code>http://www.dogpile.com/info.dogpl/search/web/haystack/<u>1</u>/-/<u>1</u>/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/417/top</code>
Page 2	<code>http://www.dogpile.com/info.dogpl/search/web/haystack/21/20/2/-/0/-/1/1/off/-/-/-/on7%253A1142948788129/-/0/417/top/-/Moderate/0/1</code>
Yahoo.com (search for “piggy bank”)	
Page 1	<code>http://search.yahoo.com/search?p=piggy+bank&sm=Yahoo%21+Search&fr=FP-tab-web-t&toggl=1&cop=&ei=UTF-8</code>
Page 2	<code>http://rds.yahoo.com/_ylt=A0S07EP_PSEmDUEmB1KNyoA/SIG=1a35brn13/EXP=1143035775/*http%3a//search.yahoo.com/search%3fp=piggy%2bbank%26sm=Yahoo%2521%2bSearch%26toggl=1%26ei=UTF-8%26xargs=12KPjglhVSt4GmmvmnCOObHb%255F%252Dvj0Z1pi3g5UzTYR6a9RL8nQJDqADN%255F2aP%255FdLHL9y7XrQ0JOKvqV2H0s3qODiIxxSdWH8UbKsmJSS%255FIppC7fdaXl1z04EdhLu3xdZvcEwdd1%252DCKIGrnZrMAebJ%26pstart=6%26fr=FP-tab-web-t%26b=11</code>
Page 3	<code>http://rds.yahoo.com/_ylt=A0Je5rSsKClEQEMRmNzXNyoA/SIG=1ee34115r/EXP=1143641644/*http%3a//search.yahoo.com/search%3fp=piggy%2bbank%26sm=Yahoo%2521%2bSearch%26toggl=1%26ei=UTF-8%26xargs=12KPjglw1SrYe9mvinCOObHb%255F%252Dvj0Z1pi298gfUcw7Ctdb8wZsHdFKaMee27khE7c73zzVzoPFrx41LLvKhaK6UAbWdU%255F9KP537Zco%255Fb1Ifv7yHZF8ny4dx2bJhNJ%252DxEEnWUSzg0%255FG96gL9PZrI51h56MlamFqDYcL67GOa3P8trJbq2yZ4717PQqtGvrUp4fyPAXY%252DU0xkFKiODxZKvMb6%252Dd1zhe%252DiJlwigB%252DQ5WgZqFw2DDMLDZ3Wa14660bkmfuYSNPN%26pstart=11%26fr=FP-tab-web-t%26b=21</code>

Table 4.1. The URL spaces of web sites are diverse and opaque. They present difficulties in determining the “next page” link given a page within a sequence of search result pages. URL fragments that vary from page to page are in bold typeface. Fragments that might specify page numbers or starting item indices are underlined.

4.2.3 Field Detection

Field values within the DOM fragments of items can be picked out if what vary among the fragments can be isolated from what are common among many of them. To detect what are common and what vary, any tree alignment algorithm [54, 63, 71] can be applied to these DOM fragments. The result of the tree alignment is a *DOM tree template* with some nodes marked as variable and some nodes marked as common, or non-variable. The non-variable nodes are decorative elements, and the variable nodes correspond to fields. Generic field names (e.g., “field0”, “field1”) are then assigned to the variable nodes in the DOM tree template.

After field assignment is complete, the template is ready to be used. The DOM fragment of each item to be extracted is aligned to the DOM tree template. If a node in the item is aligned to a variable node in the template, its text content is extracted and stored in the corresponding field.

4.2.3.1 Embedded, Typed Fields

Often, a field value does not take up a whole HTML text node. That is, you might not find a price field value “14.99” to be the only text inside a DOM text node. Instead, you might find that the text node contains “You save \$14.99!” To handle such cases, the tree alignment must work not at the granularity of HTML nodes, but at the granularity of *tokens* within text nodes.

Care must be taken when parsing “You save \$14.99!” into tokens lest “14” and “99” be split into different tokens. Splitting “14” and “99” apart will cause two different fields to be generated; sorting or filtering by a single price field is no longer possible. Similarly, splitting “Published on March 29, 1998” into tokens must keep “March”, “29”, and “1998” together as a single date token. In other words, tokenization must generate typed tokens for numbers and dates so that sorting and filtering on the corresponding fields will be possible.

A side effect of tokenization is that paragraphs of text (e.g., product review comments, publication abstracts) can cause too many fields to be generated as their tokens don’t align. To solve this problem, in the field assignment phase, if there are too many variable tokens that are siblings of one another, then their immediate parent is assigned a field name instead.

4.3 Evaluation

Sifter has been tested for robustness in its extraction algorithm as well as for usability in its user interface. The two evaluations are presented separately below.

4.3.1 Evaluation of Data Extraction

Sifter’s extraction algorithm has been tested on 30 collections of items from 30 common web sites, including Amazon, Best Buy, CNET Reviews, Froogle, Target, Walmart, and Yahoo Shopping (Table 4.3). These collections contain from about 25 to 450 items each, spanning from two to 25 pages, with each page containing from nine to 50 items. The results are shown in Table 4.4.

The item detection algorithm worked perfectly on the first pages of 23 collections (out of 30, 77% accuracy). For the failure cases, Sifter’s item highlighting tool was used and four cases were corrected. Overall, item XPaths could be found for 27 of the 30 collections (90%). In the remaining three cases, items consisted of sibling nodes, which we do not currently handle.

Among the 30 sites, 24 (80%) displayed sequences of page numbers, also called page tables of content (rather than just “Next Page” links). The Link Label Heuristic detected 13 sequences, and the URL Parameter Heuristic detected two ($(13 + 2)/24 = 63\%$). The Next Page Heuristic (requiring users’ intervention) worked on 9 of the remaining 15 cases ($9/15 = 60\%$). Overall, subsequent pages could be identified for 23 out of 30 collections (77%).

There were 21 collections for which item XPaths could be found and subsequent pages could be identified accurately. Out of these 21 collections, 19 were perfectly extracted, yielding precisely the original numbers of items. The overall accuracy of extraction is $19/30 = 63\%$.

Note that accuracy was measured per collection rather than per item as in other data extraction work. To put this in perspective, the latest work on data extraction [78] processed 72 *manually provided* pages from 49 sites and achieved $40/49 = 82\%$ collection accuracy. Over the 23 collections for which subsequent pages could be identified, our algorithm processed 176 pages automatically and achieved $19/23 = 82.6\%$ collection accuracy.

The fields extracted that could be useful for filtering and sorting included: current price, original price, percent saving, author, artist, medium, date, shipping option, brand, number of store reviews, number of bids, container size, city, etc.

4.3.2 Evaluation of User Interface

Since augmentation of web sites is a novel concept even to experienced web users, a formative evaluation of Sifter’s user interface has been conducted to determine whether it was basically usable and useful, assuming the automatic extraction algorithm performed its best.

4.3.2.1 *Design and Procedure*

This study consisted of a structured task (during which the subjects took simple steps to familiarize with Sifter) followed by an unstructured task (during which the subjects employed their own knowledge of Sifter for problem solving).

At the beginning of each study session, the subject was told that she would learn how to use something called Sifter herself but was given no particular instructions on how to use it. This was patterned after the study on the Flamenco system in which the subjects were not introduced to the system in order to better mimic real world situations [77].

Task #1 required the subject to:

- follow a sequence of simple steps to use the Sifter pane to sort and filter a collection of 48 items spread over 3 web pages obtained by searching Amazon.com for “jeffrey archer.” The desired final result was the sub-collection of only hardcovers and paperbacks published in 2004 or later, sorted in descending order by their used & new prices. The sequence of steps consisted of high-level “filter by date” and “sort by price” instructions, not low-level UI “click this button” and “select that list item” actions.
- use the Sifter pane by herself to obtain the list of 3 cheapest (if bought used) paperbacks by John Grisham in 2005 from Amazon.com.
- spend no more than 5 minutes using only the Amazon web site, but not Sifter, to find the 3 cheapest (if bought used) hardcovers by John Grisham in 2004.

Task #2 required the subject to:

- use the Sifter pane to decide whether the sale on Prada products on Ashford.com was good or not.
- use the Sifter pane to list 2 or 3 products among those that the subject considered good deals.
- use only the Ashford.com web site to judge whether the sale on Gucci products on Ashford.com was good or not, using the same criteria that the subject had used in judging the sale on Prada products.

Amazon.com was chosen for Task #1 as its search results were very structured, containing many fields useful for filtering and sorting. Furthermore, Amazon.com is popular and the subjects were more likely to be familiar with it. Ashford.com was chosen for Task #2 as its search results contained only two fields (price and percent saving), making it simpler to perform the high-level task of judging its sales.

At the end of the session, the subject rated her agreement/disagreement with 12 statements (on a 9-point Likert scale) regarding her experience learning and using Sifter.

4.3.2.2 *Participants*

Eight subjects (4 male, 4 female) were recruited by sending an e-mail message to a mailing list and posting paper ads around a local college campus. Six were in their 20s, the other two were 30s and 40s. All 8 subjects used the Web almost everyday, and all subjects visited Amazon.com at least a few times a month. None had ever visited Ashford.com.

4. EXTRACTING DATA

	Web Site	Collection within Web Site	#Items/Page x #Pages	Has Page TOC?	Total #Items
1.	acehardware.com	Kitchen faucets (plumbing)	9 x 5	●	44
2.	adesso.us	Keyboards	16 x 2	○	26
3.	alibris.com	Search for "John Grisham"	25 x 4	●	96
4.	amazon.com	Search for "Jeffrey Archer"	16 x 3	●	48
5.	ashford.com	Prada products	9 x 7	●	59
6.	bargainoutfitters.com	Women's footwear	12 x 25	●	290
7.	bestbuy.com	Point & shoot digital cameras	25 x 5	●	111
8.	buy.com	Box sets	12 x 6	○	72
9.	cameraworld.com	SLR lens over \$400	25 x 4	●	95
10.	reviews.cnet.com	Dell desktops	10 x 10	○	93
11.	compusa.com	Search for "hard drive"	20 x 15	●	287
12.	dealtime.com	Lavender (flowers and plants)	21 x 9	●	179
13.	drugstore.com	Hand creams (lotions)	15 x 6	○	87
14.	antiques.listings.ebay.com	Globes (maps, atlases, globes)	50 x 9	●	444
15.	essentialapparel.com	Women's sportswear	10 x 6	○	55
16.	froogle.google.com	Search for "rebel xt"	10 x 10+	●	>100
17.	newegg.com	Notebooks/laptops, \$1500-\$2000	20 x 3	●	42
18.	nextag.com	Hotels in Boston, 3*	15 x 4	●	58
19.	nordstrom.com	Women's wallets & accessories	21 x 4	●	74
20.	officedepot.com	Fully adjustable chairs	10 x 6	●	55
21.	overstock.com	Coins & stamps (collectibles)	24 x 3	○	58
22.	radioshack.com	All MP3 players & ipods	10 x 5	●	50
23.	rochesterclothing.com	Casual pants	20 x 3	●	53
24.	shoebuy.com	Adidas, womens, 6.5	12 x 3	●	26
25.	shopping.com	Stainless steel rings < \$50	30 x 8	●	236
26.	smartbargains.com	Ties (men's apparel)	16 x 2	●	28
27.	target.com	Clearance "table"	20 x 14	●	276
28.	tigerdirect.com	Digital photo printers	10 x 4	●	38
29.	walmart.com	Houseware, \$20-\$50	20 x 4	●	76
30.	shopping.yahoo.com	PDA cell phones	15 x 15	●	222
					3378

Table 4.3. Thirty popular retailer web sites were used to test Sifter's web data extraction algorithm. A search or browse operation was performed on each site and the number of result items per page and the number of result pages were noted. Whether the first search result page contained a table of content of links to subsequent result pages (as opposed to "Next" links) was also recorded as it affected the algorithm.

Pages Detected? - Correctable By Users?					Total Original #Items		
Items Detected? - Correctable By Users?				Total #Items Extracted - Perfect extraction?			
Has Page TOC?				Useful Fields			
Web Site							
1. acehardware.com	●	●	●	44	44	✓	price
2. adesso.us	○	●	○○	26	16		
3. alibris.com	●	●	●	96	96	✓	author, old price, new price, quick buy price, signed copies?
4. amazon.com	●	●	●	48	48	✓	author, type, date, total # items, buy new original price, buy new current price, used & new price, shipping info
5. ashford.com	●	●	●	59	59	✓	price, percent saving
6. bargainoutfitters.com	●	●	○○	290	290	✓	price
7. bestbuy.com	●	○○	●	111	0		
8. buy.com	○	●	○○	72	72	✓	artist, price, saving
9. cameraworld.com	●	○●	○○	95	95	✓	price
10. reviews.cnet.com	○	●	○○	93	93	✓	price
11. compusa.com	●	●	○○	287	287	✓	brand
12. dealtime.com	●	●	○○	179	179	✓	# store reviews
13. drugstore.com	○	○●	○○	87	15		price, size
14. antiques.listings.ebay.com	●	●	●	444	444	✓	# bids, price
15. essentialapparel.com	○	●	○○	55	110		old price, current price
16. froogle.google.com	●	●	●	>100	100	✓	minimum price
17. newegg.com	●	●	○○	42	20		shipping fee, # reviews
18. nextag.com	●	●	●	58	55		city, zip code, price 1, price 2
19. nordstrom.com	●	●	●	74	74	✓	price
20. officedepot.com	●	○○	●	55	0		
21. overstock.com	○	●	○○	58	24		
22. radioshack.com	●	●	●	50	50	✓	has ratings?, out of stock/in store/on-line, 1-2 or 2-3 days shipping
23. rochesterclothing.com	●	○●	●	53	52		price
24. shoebuy.com	●	○●	○○	26	26	✓	on sale?, price, % off, original price
25. shopping.com	●	○○	○○	236	0		
26. smartbargains.com	●	●	○○	28	28	✓	retail value, our price, % saving, # left
27. target.com	●	●	●	276	276	✓	list price, our price, save amount, save percent, shipping info
28. tigerdirect.com	●	●	●	38	38	✓	price, in stock or shipping info, brand
29. walmart.com	●	●	○○	76	76	✓	
30. shopping.yahoo.com	●	●	○○	222	210		exact price
		77%, 90%	43%, 77%	3378	2820		

Table 4.4. The results of testing Sifter's web data extraction algorithm. Whenever the algorithm failed, user intervention was attempted and the intervention's success was recorded. User intervention brought the overall success of locating items to 90% and of locating subsequent pages to 77%. The success rate for recovering *whole* collections was $19/30 = 63\%$.

4. EXTRACTING DATA

All subjects had used the Web for more than just shopping. They had searched for some combinations of the following types of information: news; images; contact information of people and organizations; maps and driving directions; hobbies (e.g., recipes, chess); reviews (on restaurants, movies, books, products); tutorials (e.g., languages, logics); and professional research (e.g., publications, scientific data).

4.3.2.3 Apparatus

Subjects received \$10 each for participating in a 30 – 45 minute study session. All sessions were conducted by one investigator on a single computer (Pentium 4 2.53GHz, 1.00GB) with an 18" LCD flat panel at 1600×1200 resolution in 32-bit color and a Microsoft Wireless IntelliMouse Explorer 2.0 (with mousewheel), running Microsoft Windows XP. UI events were recorded in a timestamped log and the investigator observed the subjects and took written notes.

4.3.2.4 Results

All subjects completed the parts of Task #1 involving Sifter. Only 5 out of 8 completed the parts involving using the Amazon web site *without* Sifter. The other subjects could not learn how to use Amazon to perform sophisticated queries within 5 minutes. Among the 5 subjects who succeeded, only one made use of Amazon's Advanced Search feature. The other 4, despite their previous experience with the Amazon web site, could only sort the items by one criterion and manually scan the list for items satisfying the other criteria. This indicates that advanced browsing features implemented by the web browser in a unified manner across web sites may be more discoverable, learnable, and usable than those same advanced features officially supported by individual web sites but have been suppressed in favor of more commonly used functionality.

Seven subjects completed Task #2 and one refused to finish Task #2 as he said he had no knowledge of Prada and Gucci products and thus could not judge their sales. For the first part of Task #2, 6 out of the 7 subjects used Sifter to look at the distribution of the percent saving. One subject could not understand how Sifter would help her judge the sale.

Table 4.2 shows encouraging evidence that the subjects found Sifter powerful yet easy to learn and use. However, the extraction process was thought to be slow. Data extraction speed depends on network performance and web server responsiveness, but on average, each test collection of 50 or so items took 30 seconds to extract.

Although there was no show-stopper problem with the user interface, some users were taken aback by the verification step (when the system announced its estimate of the items to be extracted and asked for confirmation). As they saw no other choice except clicking "Continue," they did so just to see what would happen next, in hope but not certain that that route would ultimately allow them to sort and filter. This behavior was not a surprise as web site augmentation was a new experience and the necessity for extracting data before augmentation could take place was poorly understood, if understood at all. To fix this problem, the accuracy of the algorithms must be boosted, subsequent pages must be pre-loaded and pro-

cessed even before the user clicks “Filter/Sort Items,” and the user must be allowed to make corrections from the augmentation UI.

Sorting operations took very little time and the re-shuffling of items inside the web page was too fast and subtle to shift the user’s attention from the Sifter pane where she just invoked a sorting command to the web page. The user often had to double-check the resulting list of items herself. To fix this, we can slow down or animate the changes in the web page. Filtering operations produced more drastic changes and did not suffer from the same problem.

One subject opened too many browsing control boxes and became confused as to which field each box corresponded to. He was not able to notice the synchronized highlighting of browsing control boxes and field asterisks. To fix this, we can color-code the asterisks and the browsing control boxes as well as use a variety of shapes rather than just asterisks.

Asked to filter for only items published in 2005, some subjects had to manually find one sample item published in 2005 in order to click on the asterisk next to its publishing date. Other subjects simply clicked on the asterisk next to any publishing date. If we are able to derive meaningful field names, this problem will be resolved.

Five of the 8 subjects interacted with the faded areas of the web pages when they needed to use the web sites’ functionality (e.g., performing a search for “john grisham”). The other three subjects either refreshed the web pages or retyped their URLs. In the future, Sifter’s UI can be changed to let users feel more comfortable making use of the original web sites’ features, knowing the difference between those original features and the added functionality.



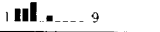

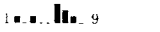
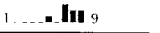

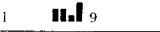

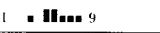
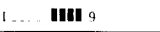

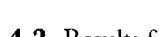
strongly disagree 1  9 strongly agree	
	1. Sifter is hard to learn how to use.
	2. Sifter is tedious to use.
	3. The filtering and sorting features in Sifter are slow.
	4. Sifter shows redundant information (easily found on the Web sites).
	5. After clicking “Continue,” I need to wait for a long time before I can use Sifter.
	6. Sifter is simple to use.
	7. Sifter is powerful (providing advanced features).
	8. Sifter displays interesting information.
	9. Sifter displays useful information.
	10. Sifter is enjoyable to use.
	11. Sifter adds value to the Web sites in this user study.
strongly disagree 1  9 strongly agree	12. I believe Sifter will add value to some other Web sites I have used.

Table 4.2. Results from the exit survey of the formative evaluation show encouraging evidence that the Sifter pane is usable (#1, #2, #6) and useful (#8, #9, #11, #12) even when it is considered to offer advanced functionality (#7).

4. EXTRACTING DATA

One subject—the only one who used Amazon’s Advanced Search feature—asked when she would be able to use Sifter in her own browser. She mentioned that there were a number of sites she used frequently which did not offer the browsing functionality that she needed. Another subject said that although he never had to perform the kinds of task in this study on the Web, he had to perform similar tasks in spreadsheets.

4.4 Summary

This chapter puts forth ideas in which presentational elements of web pages can be retained and used in web data extraction toward saving users from the tedious field labeling task as well as enhancing the usage of the extracted data. Advanced browsing features can be added right inside the original pages, preserving visual context and leveraging custom presentational designs on the original pages rather than downgrading to a generic presentation template.

5. INTEGRATING DATA

As demonstrated in the last chapter, it is feasible today to let casual users with no programming skills extract data from the text-centric Web for reuse. Such capability is already useful for any single web site that does not offer the features that a casual user needs. More empowering to casual users is the ability to combine data from several sites to get values that no single site can offer by itself.

Like conventional data extraction tools, conventional data integration tools have also been built for a different audience than casual users. For a typical conventional data integration task, such as merging huge databases of two institutional libraries together to serve the combined data through a new web site, several experts and programmers are involved to handle different aspects of the task, including aligning the schemas, cleaning up the data, and then building the web site. Each tool employed is specialized for one stage of the process and designed to handle large, complex data sets. In contrast, a casual user might just want to merge two lists of a few dozens of addresses from two web sites together to plot them on a common map. There is much less data and the data is much simpler. Power tools used by experts are too advanced for casual users.

Compared to experts, casual users lack both data modeling skills and programming skills. However, for small, simple data sets, neither skill set may be crucial.

- First, when the data set is small, schemas—useful for efficiently reasoning about a massive quantity of data in the abstract—are not as useful and can introduce overhead cognitive costs. Instead of having to learn about theoretical concepts like schemas, casual users can manipulate data instances directly. Visual semantics are often enough: if the data “looks right” in the user interface, it most probably has been edited correctly and there is no need to verify the data model.

- Second, instead of using programming to process data, casual users can just use direct manipulation techniques. For example, two fields can be aligned by dragging and dropping one onto the other.

These ideas have been built into Potluck, a tool that lets casual users—non-programmers—integrate data all by themselves. This chapter will next describe a hypothetical usage scenario for Potluck. Using various challenges in that scenario as motivations, the user interface of Potluck will be explained. Then Potluck’s implementation will be detailed. Finally, an evaluation of Potluck is reported.

5.1 Scenario

Before describing the user interface of Potluck, this section motivate it with a scenario that illustrates various idiosyncrasies of data integration. Let us be optimistic that within a decade, the Semantic Web will be prevalent and RDF data will be everywhere. Even in this future world, users will *still* face problems integrating data from different sources and tools such as Potluck are still needed.

In 2017, a historian named Henry is documenting the first cases of a rare genetic disease called GD726. These first cases occurred in the Valentine family in the 1820s. He wants to include in his final report a genealogical tree of the Valentine family, annotated with the disease’s infliction, as well as a comprehensive table of the Valentines’ data in an appendix.

Like most historians, Henry is not a programmer but he is experienced in collecting and managing data in his professional work. The proliferation of RDF means that Henry does not need programming skills to scrape HTML himself: all the information needed for his research has been converted into RDF by various independent organizations and individuals, both professionals and enthusiasts. Henry thinks it would be trivial to simply pool the RDF together and call it done.

Henry tracks down various birth certificate issuing offices and death certificate issuing offices where the Valentines lived for their RDF data. He notes that some offices use `dc:date` in their data to mean “birth date,” some to mean “death date,” and some “certificate issuing date.” It would be disastrous to consider all the `dc:dates` the same even if the same predicate URI is used.

Henry also tracks down hospital records, which contain `hospital:tod` (short for “time of death”). Hence, `hospital:tod` is equivalent to some of the `dc:dates`. It would be hard to match `hospital:tod` with `dc:date` based on string analysis alone, yet match for some of the cases only.

The records all have geographical location names, but these names are not fully qualified. Those responsible for digitizing them thought that since all locations were within their country, there was no need to include the country name. As a consequence, Henry needs to append the country name to the many location names in order to map them.

People’s names are encoded in two different forms: “first-name last-name” in some data sets and “last-name, first-name” in others. Nick names are also present (e.g., “Bill” instead of “William”, and “Vicky” instead of “Victoria”).

The hospital records also pose problems. While most of their admittance dates are in ISO 8601 format, a few are of the kind “Easter Day 1824.” Such sloppiness has been observed in industrial and institutional databases, and should be expected on the Semantic Web.

Despite all these problems, there is one good thing about the data: Henry can reliably get the mother and father of each Valentine through the `gen:mother` and `gen:father` predicates, which seem to be very widely adopted. This helps Henry construct a genealogical tree visualization.

However, as males and females both have equal chance of passing on GD726, Henry wants to treat `gen:mother` and `gen:father` the same while tracing the disease through the family. Unfortunately, adding an `owl:sameAs` equivalence between those two predicates will break his genealogical tree.

While all parties involved in this scenario acted logically and responsibly, Henry still ends up with a mess of RDF. To fix up the data, Henry must be able to:

- Merge `dc:dates` into *several* groups (the birth dates and the death dates) even though they all use the same predicate URI. This requires distinguishing the fields by their origins rather than just by their URIs.
- Merge `gen:mother` and `gen:father` together in some situations while keeping them separate in other situations. This precludes the simple approach of adding `owl:sameAs` statements in the data model to implement equivalences.
- Edit the data efficiently to unify its syntax.
- Fix up the data iteratively as he learns more and more about the data.

These are the tasks that must be supported by such a tool as Potluck in order for a casual user such as Henry to be able to integrate data all by himself.

5.2 User Interface

This section describes Potluck’s user interface, showing how it addresses the problems in the scenario above. The reader is encouraged to view this screencast to understand Potluck’s interactivity:

<http://people.csail.mit.edu/dfhuynh/research/media/iswc2007/>.

Figure 5.1 shows the starting screen of Potluck where the user can paste in several URLs and click **Mix Data**. This results in Figure 5.2, which lists data records from the original web pages. The records are interleaved by origins—the pages from which they have been extracted—to ensure that some records of each data set are always visible.

Fields are rendered as *field tags*: `label`, `position`, and `title`. Field tags are color-coded to indicate their origins: blue from one source and pink from another in

5. INTEGRATING DATA

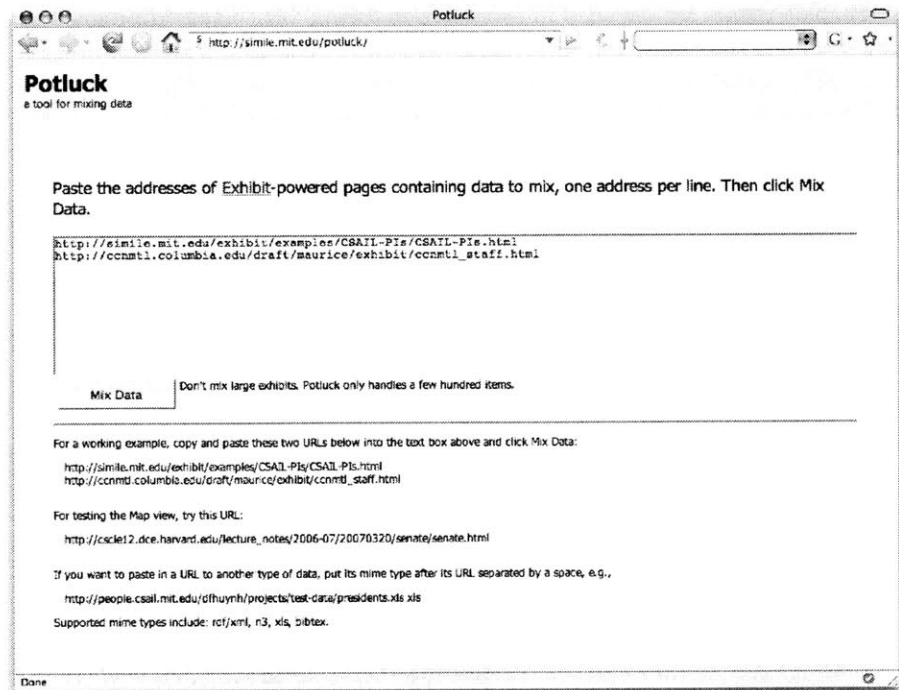


Figure 5.1. The starting screen of Potluck. Clicking **Mix Data** yields the mixed data in a screen like Figure 2.

Figure 5.2. Three core fields, **label**, **type**, and **origin**, are automatically assigned to all records and their tags are colored gray. Fields from different origins having the same name are considered different. For example, while **phone** means office phone, **phone** might mean secretary's phone. Or more dangerously, **dc:date** in the scenario (in section 2) has several distinct meanings. These semantic differences, subtle or significant, might or might not be important to one particular user at one particular moment in time. Keeping the fields apart rather than automatically merging them together allows the user to make the decision whether or not to merge.

The screenshot shows the Potluck web application interface. The browser address bar displays `http://simile.mit.edu/potluck/system/template/browse`. The page title is "Potluck" and the data origins are listed as "MIT CSAIL Faculty web site" and "CCNMTL Staff web site". There are "Export tsv" and "Export json" buttons in the top right. The main content area is titled "VIEW AS: TABLE • MAP" and includes options to "condense details", "expand all details", and "collapse all details".


The interface is divided into several sections:

- New Column:** A sidebar on the left with the instruction "drag and drop fields to create columns".
- Details:** A central area showing details for three individuals:
 - 1. Alan Edelman:** label: Alan Edelman; origin: MIT CSAIL Faculty web site; email: edelman@csail.mit.edu; floor: 6th floor; group: unknown; last-name: Edelman; office: 32-G606; phone: 253-1355; photo: <http://www.csail.mit.edu/P1/72dpi/edelman.jpg>; position: faculty; tower: Gates; url: <http://www.csail.mit.edu/~rint.php?PeopleID=75>; type: Person.
 - 2. A. Maurice Metz:** label: A. Maurice Metz; origin: CCNMTL Staff web site; bio: Maurice is a co-founder of the Columbe Center for New Media Teaching and Learning and currently pla... ('82) in Computer Science from the School of Engineering and Applied Science at Columbia University.; building: Butler Library; campus: Morningside; email: maurice@columbe.edu; first: Maurice; group: Management; Technology; imageURL: <http://ccnmtl.columb.edu/headshots/Metiz.jpg>; last: Metz; phone: 212 854-2665; room: 505A; title: Vice Executive Director, Director of Technology; type: Staff.
 - 3. Alan Willisky:** label: Alan Willisky; origin: MIT CSAIL Faculty web site; email: willisky@mit.edu; floor: 5th floor.
- Facets:** A sidebar on the right with the instruction "drag and drop fields to create facets". It shows a facet for "origin" with two items: "33 CCNMTL Staff web site" and "92 MIT CSAIL Faculty web site".

The bottom of the browser window shows "Done".

Figure 5.2. Potluck’s user interface shows data that has just been mixed together but not yet processed by the user. Fields are rendered as draggable “field tags,” color-coded to indicate their origins. There are two drop target areas for creating columns and facets.

5.2.1 Creating columns and facets

A field tag can be dragged and dropped onto the gray column to the left (Figure 5.2) to create a new column listing that field, or onto the gray box to the right to create a facet for filtering by that field. Figure 5.3 shows a newly created column. A column or facet can be moved by dragging its field tag and dropping the tag between other columns or facets. Deleting a column or facet (by clicking its ) removes the column or facet from the display but does not delete the corresponding field's data.

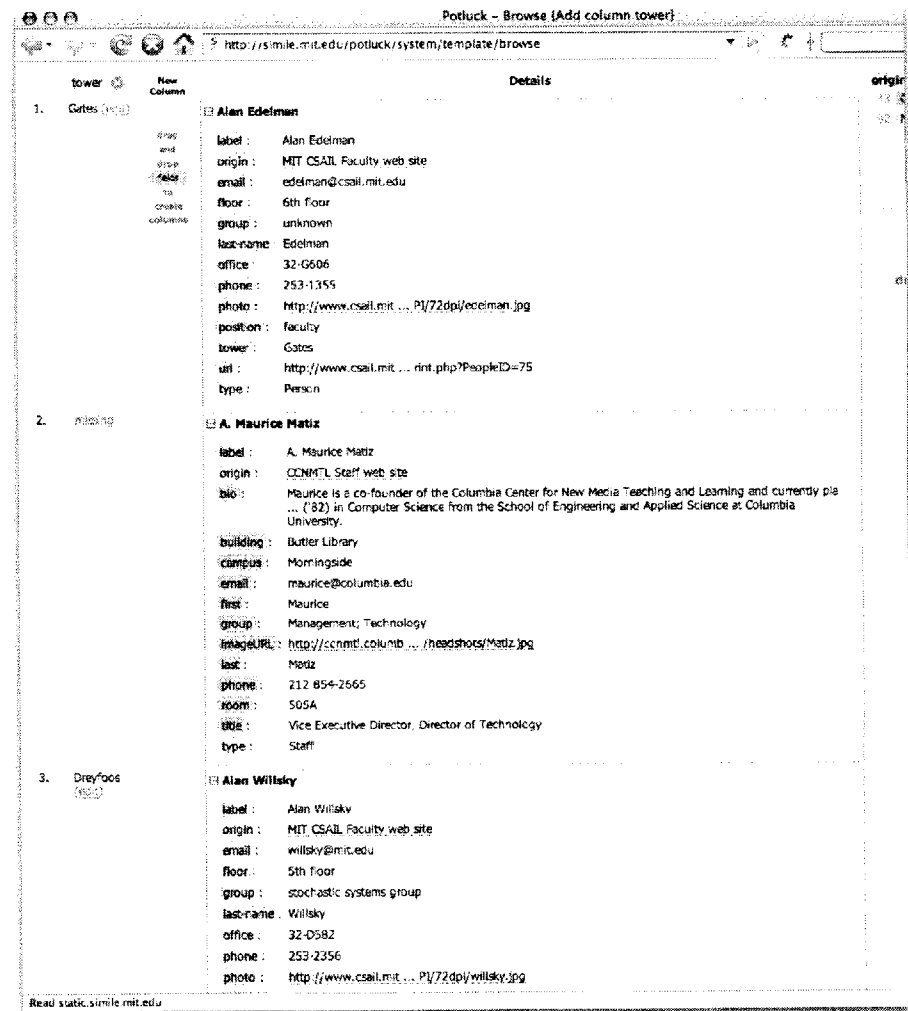


Figure 5.3. Potluck renders a new column to the left when `tower` is dropped into the New Column drop target. Since the second record is not from the same origin as the dropped field, its cell in that column shows `missing`.

5.2.2 Merging fields

A field tag can be dropped onto an existing column or facet in order to make that column or facet contain data for both the original field and the newly dropped field. Such an operation creates a *merged field*, whose field tag is rendered as a visual juxtaposition of the original tags, taking on a pill-shaped form `position title`. Figure 5.4 shows several columns and facets of merged fields. Merged field tags can be dragged and dropped just like elemental field tags can in order to create new columns and facets, or to merge into other existing columns and facets.

Creating a merged field does not disturb the elemental fields. Thus, in the scenario, it would be easy to have `gen:mother` and `gen:father` merged together for one purpose while keeping them separate for another purpose, all at the same time.

The screenshot shows the Potluck web application interface. The browser address bar displays `http://simile.mit.edu/potluck/system/template/browse`. The page title is "Potluck" and the data origins are listed as "MIT CSAIL Faculty web site" and "CONMTL Staff web site". The interface includes a search bar with the text "You've just done: select 'Gates'" and a "VIEW AS: TABLE + MAP" option. The main content area displays a table with columns for "tower_building", "photo imageURL", "email | email", and "phone | phone". The table contains five rows of data, each with a merged field tag in the "tower_building" column. To the right of the table is a "Details" panel showing the details for the selected record, "Alan Edelman". Below the table is a "Facets" panel showing a list of categories and their associated records, including "origin", "tower_building", and "group | group".






Record ID	tower_building	photo imageURL	email email	phone phone
1.	Gates		edeiman@csail.mit.edu	253-1355
2.	Lewisohn Hall		acox@cnmtl.columbia.edu	212 854-1851
3.	Gates		meyer@csail.mit.edu	253-5024
4.	Lewisohn Hall		fnorvati@columbia.edu	212 854-1692
5.	Gates		agarwal@csail.mit.edu	253-1448

Figure 5.4. A screen shot of Potluck showing several columns and facets of merged fields. The records' details have been collapsed to make space for the columns.

Furthermore, the merging operation is *not* transitive, so that, say, merging fields **mother** and **father** together (to mean **parent**) and then **mother** and **grandmother** together (to mean **female ancestor**) does *not* force all three fields to be merged into **mother/father/grandmother**.

5.2.3 Simultaneous editing

The **edit** link next to each field value opens up the Simultaneous Editing dialog box where the values of that field can be edited *en masse* (Figure 5.5). The concept of simultaneous editing originated from LAPIS [59], a text editor that displays several keyboard cursors simultaneously on a text document, generalizes the user's editing actions at one cursor, and applies them to the text at the rest of the cursors. Based on the user's mouse clicks, LAPIS guesses how to divide the text document into records (often into lines or paragraphs) and where the cursors should be placed within those records (e.g., after the second word of the third sentence in each paragraph). Whereas LAPIS has to guess what a record is for the purpose of simultaneous editing, Potluck already has the field values conveniently separate. Potluck groups field values into columns by structural similarity, e.g., the phone numbers in the second column all have area code 212. These columns serve to visually separate out values of different forms, call out outliers (such as "Easter Day 1824" in the scenario), and let the user edit different forms differently. The user can click on any field value to give it keyboard focus, and editing changes made to it are applied to other values in the same column in a similar fashion. The multiple cursors in Figure 5.5 give visual feedback of the simultaneous editing operations in progress.

If a value appears in several records it is shown in only one entry in the dialog box. In the scenario, if the nickname "Bill" appears in three records, the user can

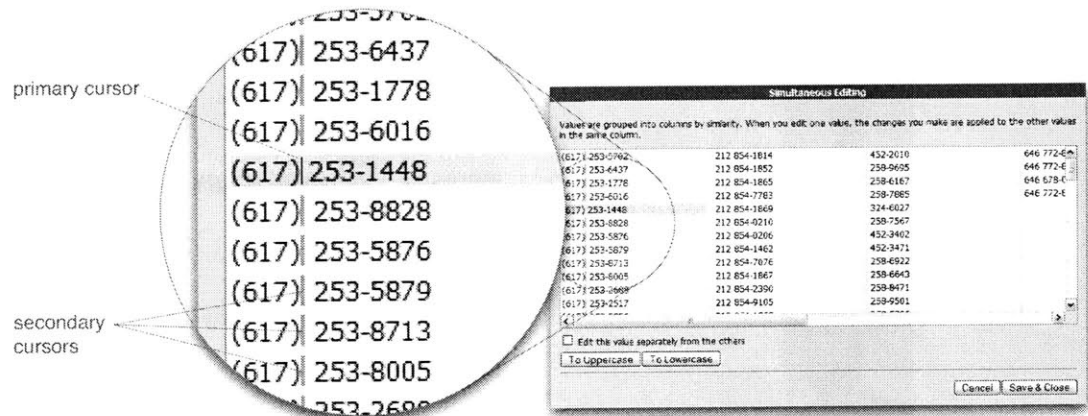


Figure 5.5. Potluck's Simultaneous Editing dialog box lets the user change several similar values simultaneously by editing any one of them. Multiple keyboard cursors are shown and any editing change to the focused value is immediately reflected in the other values.

click on its single entry in the dialog box, set the checkbox **Edit this value separately from the others**, and change it to “William” to correct all three records.

Simultaneous editing is useful for correcting inconsistencies between data sets that occur many times, such as prefixing area codes to phone numbers and wrapping existing area codes in parentheses. It is also useful for reformatting a field, such as changing “first-name last-name” into “last-name, first-name”, and for making a new field out of an existing field, such as extracting building numbers (32) from within office numbers (32-582).

5.2.4 Faceted browsing

Faceted browsing [77] is a browsing paradigm in which a set of records can be filtered progressively along several dimensions in any arbitrary order. For example, a set of recipes can be filtered by picking an ingredient first, a cooking method second, and a cuisine finally, or by picking a cuisine first, then an ingredient, and a cooking method finally depending on which order suits the user best. Because the data Potluck handles is often multidimensional, faceted browsing is useful in Potluck as it is designed for exploring multidimensional data in flexible, user-controllable ways. Exploration is needed for identifying and selecting out just the subset of data that is useful as well as for isolating on records that need cleaning up. All faceted browsers so far work on single data sets. Potluck extends faceted browsing for the data integration task in which data arrives from many sources.

If within a facet there are records for which the corresponding field is missing, the facet explicitly shows a choice for filtering to those records (Figure 5.6). This visual element, not present in conventional faceted browsing interfaces, also serves

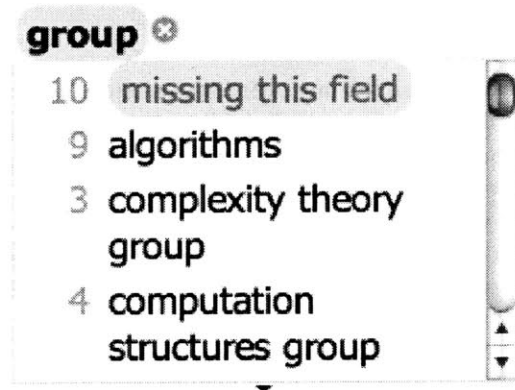


Figure 5.6. If inside a facet there are records for which the corresponding field is missing, the facet shows `missing this field` as a choice so that the user can get to those records.

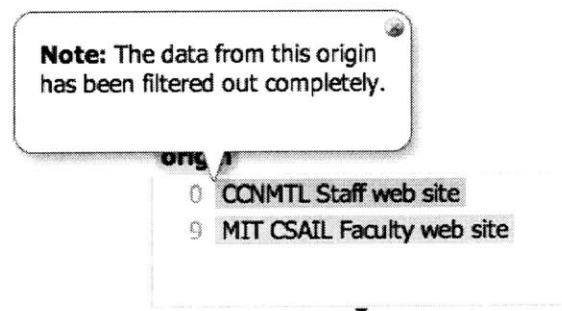


Figure 5.7. The origin facet does not remove choices for which there are no records. Moreover, it pops up messages to call the user’s attention to those filtered out origins.

to remind the user that, if that field is an elemental field instead of a merged field, the field is not present for records in other data sets.

While working with multiple data sets at the same time, it can be easy to forget that an elemental field from one data set does not exist in the others. Whenever a facet choice causes all records from an origin to be filtered out completely, that origin remains in the origin facet and a message is popped up drawing the user's attention to it (Figure 5.7).

5.2.5 Visualizations

Potluck currently provides two visualizations: a tabular view and a map view. Figure 5.8 shows the map view in which any field containing street addresses or latitude/longitude pairs can be dropped onto the map view to plot the records. The map markers can also be color-coded using drag and drop. Faceted browsing is supported concurrently so that the user can construct a map while browsing through the data at the same time.

5.2.6 Miscellany

Potluck provides drop down menus on left clicks as alternatives to drag and drop in order to increase the likelihood that the user succeeds at finding some way to accomplish a task. The browser's Back and Forward buttons can be used to redo and undo user actions. Like contemporary highly interactive web interfaces, Potluck also shows the most recently done or undone action and provides a link to undo or redo it.

5.3 Implementation

Potluck consists of two components: a server-side component implemented as a Java servlet, responsible for retrieving the data within the Exhibit-embedding web pages to mix; and a client-side component implemented in Javascript on top of the Exhibit API, responsible for all the user interface interactivity.

Merged fields are implemented as query unions: when the values of a merged field are requested, the values of each elemental field in that merged field are returned in a single result set. No equivalence is added into the data model so that merging operations will not be transitive and so that the original elemental fields can still be used in isolation even after they have been merged.

Simultaneous editing is implemented in Javascript. Each field value is parsed into a sequence of features. Features are runs of digits, of letters, or of white spaces, or individual punctuation marks and symbols. For example, "733-3647" is broken down into three features: the run of digits "733", the symbol "-", and the run

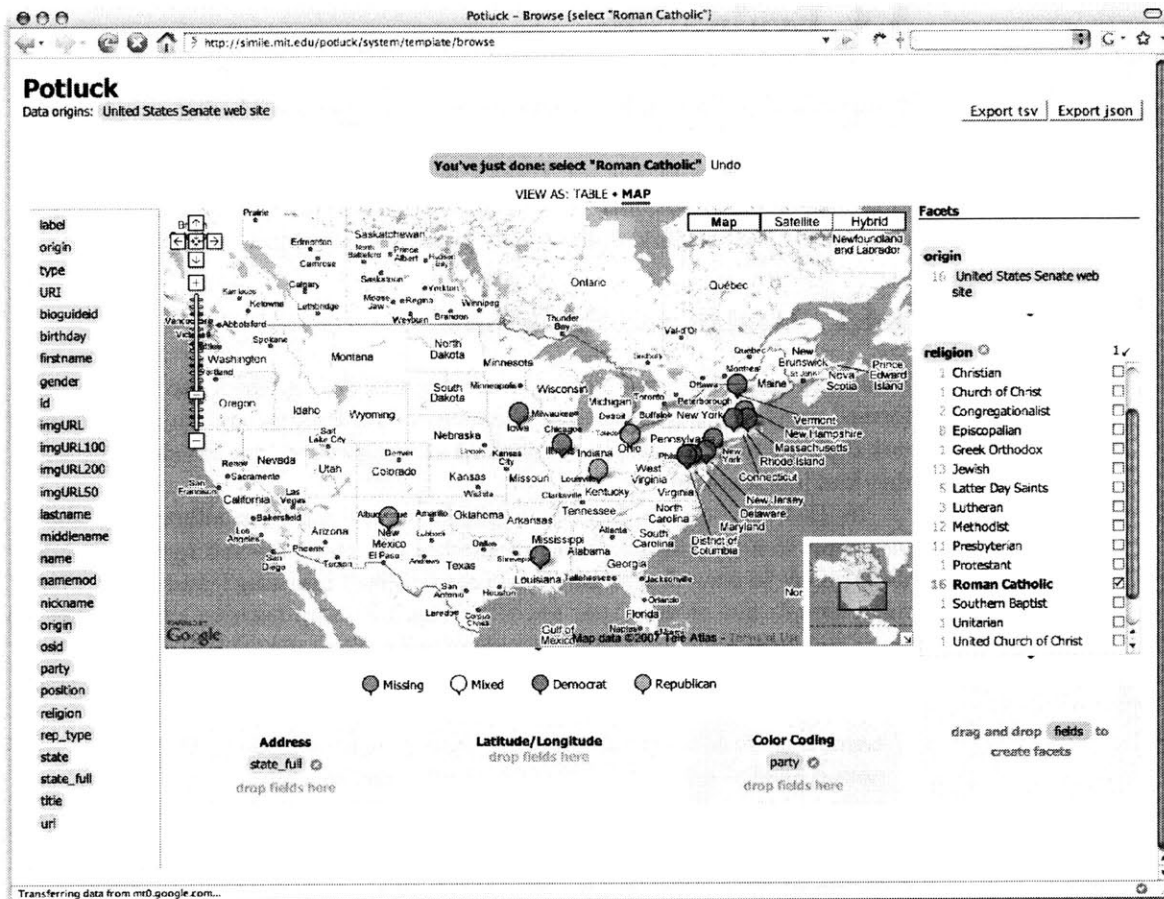


Figure 5.8. Potluck’s map view allows plotting and color-coding records by dropping field tags into drop target areas. Faceted browsing is also offered during map construction.

of digits “3647”. Field values are then clustered into columns by greedily aligning these sequences of features.

As the user moves the keyboard cursor, makes selections, and edits the text of one value, the cursor positions are generalized to be relative to the features of the field value being edited (e.g., “second character from the beginning of the third last feature”), and then those generalized cursor positions are turned into absolute cursor positions of each of the other field values in the same cluster and used to apply the edit. Secondary cursors are rendered using colored `` elements.

As the clipboard Cut and Paste operations cannot be reliably detected, cut-and-paste must be supported in simultaneous editing using a trick. When some text is inserted, if that same piece of text has previously been deleted in one edit action, it is assumed that what has taken place is a cut-and-paste operation. Note that this trick works only for cut-and-paste, not copy-and-paste.

5.4 Evaluation

A user study on Potluck has been conducted to ascertain whether people could learn how to use Potluck as well as to discover usability problems. Another purpose was to observe how people use Potluck in an open-ended task using their own judgement about which fields to merge and edit, and how to display them, so as to determine if casual users could actually perform data integration themselves.

5.4.1 Design and Procedure

This study consists of two tasks: a structured task during which the subjects performed simple steps to familiarize themselves with Potluck, and an unstructured task during which the subjects performed an open ended task based on the skills they had just acquired.

In Task #1, subjects browsed two web pages containing information about 92 people in a lab and 33 people in another lab, and answered questions about these people in ways that required the pages' faceted browsing features (e.g., "how many people are in the Gates tower?"). This warm-up exercise let the subjects learn about the data and about faceted browsing. Then the subjects were asked to use Potluck to mix the data in those web pages and to achieve the following goals (quoted almost verbatim from the study's instructions):

- create a column listing the buildings where people work and make sure the column is filled in with information for people from both labs;
- create a column listing people's phone numbers and edit them to have the form (xxx) xxx-xxxx, using 617 for phone numbers without area code;
- create a column listing people's job titles;
- create a facet of people's job titles, use it to filter for people in directing positions (directors and co-directors), and determine how many such people there are in each lab; and
- create a column of people's last names and sort it in ascending order.

These instructions were *not* worded in low-level details (e.g., click this button) so to allow the subjects the opportunities to learn how to use Potluck's user interface by themselves and to allow us the chance to discover usability problems.

In Task #2, the subjects were asked to use Potluck to mix data from two Exhibit-powered web pages of 40 + 55 publications and then mock up a single web page where hypothetical visitors could conveniently sort and filter through all of those publications as if the data came from a single source. The subjects were left to their own discretion to decide which columns and facets to create, although some examples were given in case the subjects were not familiar with the domain.

5.4.2 Participants and Apparatus

Six subjects (2 male, 4 female) from a university community were recruited by sending an e-mail message to a mailing list and posting paper ads around our college campus. Four were younger than 30, and two older than 30. They were two students (mechanical engineering and computer science), two researchers (applied math and brain and cognitive science), a lawyer, and an applied math consultant.

Five subjects (1 male, 4 female) were also recruited from a local campus' libraries, who worked with data in their daily job. Two were in their 20s, one 30s, and two 40s. There was a desire to observe if librarians, who have more experience working with data, would use Potluck differently.

There were a total of 11 subjects, referred to as G1 to G6 from the general university population and L1 to L5 from the libraries. All browsed the Web at least a few times a day and used Firefox as one of their primary browsers.

Subjects received \$10 each for participating in a 30 – 45 minute study session. All sessions were conducted by one investigator on a single computer (Pentium 4 2.53GHz, 1.00GB) with an 18" LCD flat panel at 1600×1200 resolution in 32-bit color and a Dell two-button mouse with wheel, running Microsoft Windows XP. The study facilitator observed the subjects and took written notes.

5.4.3 Results

All subjects were able to learn Potluck's user interface with little guidance and to complete the user study's tasks within 45 minutes. We now report the results in more details and point out usability issues to address in the future.

5.4.3.1 Columns

Nine subjects out of 11 used only drag and drop to create columns. This indicates that the relevant visual cues were sufficiently strong. One of the other two subjects, G5, used the Create Column menu command at first but adopted drag and drop later. L1 used only the menu command.

G5 and L5 had difficulty understanding that dragging a field tag to create a column automatically filled up the whole column with data wherever the field was available. They continued to drag the same field tag out again and again for each row, paying no attention to the data already shown in the column. The drag feedback can be improved to better indicate that the whole field is being dragged, such as showing ghosted images of several field values near the mouse pointer.

All except one subject merged columns using drag and drop; G2 used the corresponding menu command. G3 and G4 expected the phone fields from both sources in Task #1 to be merged automatically. Potluck can be made to suggest such merging if the field names match precisely.

Most subjects merged **position** and **title** together into one column, but one subject also included **group** to more fully qualify **position**. This was because most **title** values were more specific than most **position** values (e.g., "Codirector of Marketing" vs. "professor"). This operation was actually not what the subject in-

5. INTEGRATING DATA

tended (as he verbalized): the operation performed a set union of two fields instead of a string concatenation. But as Potluck rendered the **group** value after the **position** value for each record (e.g., “professor, computer architecture”), the visual outcome looked right and the subject was contented. However, sorting on this merged field would produce random orders and a facet created out of this merged field would list the **group** and **position** values separately, not paired together. Potluck should support string concatenation and suggest it as an alternative to merging whenever the two fields involved come from the same source. Note that in the scenario in section 2, concatenation is probably not the desired choice when the **gen:mother** field is dropped onto the **gen:father** field even though both come from the same source. This is why heuristics should only be used to make suggestions, not to automate.

5.4.3.2 Facets

All subjects used drag and drop to create facets. Two subjects initially created facets using the corresponding menu command, but they discovered the drag and drop alternative and did not revert to the menu. Merging facets was done solely using drag and drop. Note that the field tags on facets do not offer any menu (an oversight in our implementation); only field tags in the details column and in the column headers support menus.

Some subjects tended to drag already merged field tags from columns to create facets while the others dragged elemental field tags from the Details column to create merged facets. The latter behavior forced the user to re-merge fields she has already merged; this is both inefficient and error-prone as some subjects did forget to re-merge fields. Potluck should have automatically suggested or defaulted to the merged field whenever an elemental field that has been merged is used.

G4 did not initially merge facets in Task #1 to filter for people in directing positions. Instead, he created two facets, **position** and **title**, from the two sources separately and used `missing this field` to achieve the goal. In either facet, he selected directing positions as well as `missing this field` so that records in the other source were not excluded. This required on his part deeper understanding of how faceted browsing worked. When asked to achieve the goal without using `missing this field`, he discovered that he could merge facets.

5.4.3.3 Simultaneous editing

All subjects were able to edit several phone numbers using the simultaneous editing feature. G1 anticipated this feature even before clicking **edit**, asking out loud, “can I edit them all together?” She later used the feature to delete first names from people’s full names to get a field of last names. This action properly utilized the simultaneous editing feature’s power but destroyed data (the first names). Potluck can be made to alert the user of this loss and offer a convenient way to apply the edit on a copy of the original field instead.

G4 tried to move the leading “A” from publication titles to the end (e.g., “Tale of Two Cities, A”) using simultaneous editing (a reasonable goal) but the facilitator explained that the feature did not support that case. L2 and G6 tried to swap first

names and last names so that publications could be sorted by their authors' last names. L2 selected a last name in the simultaneous editing dialog box and dragged it to the front of the corresponding first name; unfortunately, a bug prevented this from working. G6 used keyboard shortcuts for cut-and-paste and succeeded. These subjects' actions indicated some intuitiveness in using cut-and-paste and drag-and-drop for simultaneous editing.

G3 expressed that she did not want to see all phone numbers in the simultaneous editing dialog box but only their templates. G5 and L3 edited only the first group of phone numbers, and L4 edited only the first and third groups, neglecting the groups that were not scrolled into view. To avoid such oversight, which pieces of data an edit does and does not affect must be made apparent.

5.4.3.4 *Librarians vs. general subjects*

Among the five librarians, four were catalogers (who characterize physical artifacts such as books and enter their metadata into databases), and one was a programmer responsible for integrating large data sets. While the catalogers showed no significant difference with the general subjects in their use of Potluck, the programmer, L1, was clearly an outlier: he created 10 columns and 7 facets in total. He was very excited about the user interface of Potluck and described his data integration work, consisting of manual data entry and Perl scripting, to be tedious and painful.

G6, who also needed programming skills to deal with some data for his work, expressed equal enthusiasm for Potluck. He used simultaneous editing to swap first name and last name. Thus, while there was no noticeable difference between the subjects from the general population and the librarians, who purportedly work with data and metadata on a daily basis, there was a difference between programmers and non-programmers in how much they appreciated Potluck. Programmers, who have encountered difficulties in dealing with data even with their programming skills, appreciated Potluck more. Non-programmers accomplished the tasks in the study equally well, but were not equally excited perhaps because there was not enough reusable data on the Web for them to feel the need to integrate data themselves. However, when there will be more reusable data in the future, Potluck will level the playing field for non-programmers, making them as effective as programmers for the task of integrating data.

5.5 Summary

This chapter presented several techniques embodied in a tool called Potluck for letting for casual users—those without programming skills and data modeling expertise—integrate data by themselves and obtain usefulness from the integrated data. Potluck is novel in its use of drag and drop for merging fields, its integration and extension of the faceted browsing paradigm for focusing on subsets of data to align, and its application of the simultaneous editing technique for cleaning up data syntactically. Potluck lets the user construct rich visualizations of data in-place as

5. INTEGRATING DATA

the user aligns and cleans up the data. This iterative process of integrating the data while constructing useful visualizations is desirable when the user is unfamiliar with the data at the beginning—a common case—and wishes to get immediate value out of the data without having to spend the overhead of completely and perfectly integrating the data first. A user study on Potluck indicated that it was usable and learnable, and even solicited excitement from programmers who had experienced great difficulties in integrating data even with their programming skills.

6. CONCLUSION

In this thesis I set out to explore user interfaces for supporting data-centric interactions on today's Web. In doing so, I have shown that such data-centric interactions are useful to and feasible for casual users, and usable tools can be built to support such interactions by gearing for small, simple data sets and common casual needs.

This thesis makes the following contributions:

- First, this thesis combines a simple graph-based data model with simple extensions of the HTML syntax in order to let casual users—those without programming skills—publish web pages offering advanced browsing features and rich visualizations.
- Second, this thesis shows that the cost of using web data extraction technologies can be lowered for casual users by retaining presentation elements from the original web pages. These elements preserve visual context and visual semantics so that direct manipulation techniques can be applied to augment the original pages with more features without requiring users to label fields in the extracted data.
- Third, this thesis proposes that for simple data sets, direct manipulation techniques can be used to let casual users integrate data and construct rich visualizations from it without any need for programming.
- Finally, by letting casual users publish data to the Web in browsable form, extract and reuse data from the Web, and integrate data from several sources without any need for programming, this thesis demonstrates that data-centric interactions can be offered to casual users on today's Web without having to wait for a semantic web.

This chapter continues with possible future directions from this line of investigation and concludes with a discussion on larger contexts surrounding my research.

6.1 Future Work

This section proposes future work items organized by the three core aspects of the thesis. At the end, it draws together all three aspects and proposes a way to build a data-centric browser for interacting with the future data-centric Web.

6.1.1 Publishing Data

Small-time publishers already benefit just to be able to publish 1,000 items using Exhibit. However, over time, they might develop sophistication in their publishing needs, wanting to go beyond that limit. A smooth transition path from 1,000 items to tens of thousands of items is required to migrate this class of users.

Small-time publishers will also demand easier editing interfaces. While HTML was good enough to bootstrap the Web, it was not until the arrival of blogs and wikis that user content exploded. Various user interface ideas in Potluck can be moved into Exhibit so that publishers can edit their data and their exhibit layouts directly within the browser using direct manipulations.

Users of exhibits will also want more control over Exhibit. If an exhibit is not configured in the way that best serves a user, such as not showing a map view, the user will benefit from being able to reconfigure the exhibit in-place, adding a map view. Again, Potluck's direct manipulation techniques can be adapted for this purpose.

Another future direction can explore the benefits of embedding Exhibit in online scientific publications. Currently, scientific papers are published as PDF files and the data graphics they contain are just static images. If these papers were instead published as web pages that embed Exhibit, their data graphics can be interactive and their data can easily be reused by their readers. Some technologies are needed to migrate the class of scientific publishing users from their conventional PDF publishing process over to data-centric web publishing.

If the use of Exhibit proliferates, and there are millions of data-rich web pages created by casual users, search engines should be adapted to leverage the data within these pages, providing search capabilities that go beyond keyword queries.

6.1.2 Extracting Data

Sifter lowers the cost to using web data extraction technologies by retaining much of the original presentation. But more can still be retained. For example, if used

prices are shown in a gray, stroke out typeface on the original web page, the browsing control box for used prices should also use that same typeface to display its facet values. This will strengthen the connection between a browsing control box the corresponding field values in the items.

The augmentation interface can be moved earlier in the process. For example, even before extraction, Sifter can let the user right-click on any part of the web page and invoke a sorting or filtering command, which would then trigger the extraction process and show the browsing control box for the corresponding field. The extraction over subsequent pages can also be done incrementally. That is, the augmentation interface can be shown right away and the extraction process updates it gradually as more and more pages are scraped. This change in the workflow lets the user invoke the desired feature earlier and start to get the results earlier.

Sifter should also let the user cancel the incremental extraction process or make correction at any point in time. Correction may involve, for example, splitting a field into several fields, merging several fields into one, and changing the value type of a field.

6.1.3 Integrating Data

The idea of retaining original presentation can be carried over from Sifter into Potluck so that the user will not be faced with raw data at the beginning. Research is needed on how to compose the presentations from different sources. For example, if one original site offers only a map view, and another offers only a timeline view, what should Potluck show at the beginning? While an obvious answer would be to juxtapose the views, as the views remain separate, this solution might actually discourage the user from trying to mix the data together.

Once a user has integrated data from some sources and constructed visualizations for that data, the actions she took to align the data, to fix it up syntactically, and to construct the visualizations can be stored and used to make suggestions to other users who want to deal with the same sources.

6.1.4 Toward a Data-Centric Browser

As there is more and more data in reusable forms on the Web, casual users will encounter it more often and their use of the data will increase in frequency and in sophistication. Where casual users meet the Web—the web browser—will need to catch up. Designed for the text-centric Web for viewing hypertext documents, the contemporary browser may no longer be suitable for the data-centric Web. A data-centric browser is needed to address casual users' needs in interacting with a future data-centric Web.

One of the Web's greatest strengths is in its generality. Documents on any topic can be published to the Web and viewed and interacted with in any web browser. Each web browser provides a set of basic, generic features that are sufficiently useful for browsing anywhere on the Web. Similarly, every data-centric web browser

6. CONCLUSION

should provide some set of basic, generic features sufficiently useful for interacting with any data encountered on the data-centric Web.

The chief advantage of a data-centric Web over a text-centric Web is the possibility of repurposing data—using it in ways different from how it was originally intended to be used. In order to repurpose data you must first be able to *retrieve* it, which might involve screen scraping for web pages that have not become data-centric. Then, as shown in the previous two chapters, repurposing data involves *browsing* through it in ways not supported at the original sites, *merging* data from several sites together, *editing* the data to clean it up and unify it syntactically, and *visualizing* it in novel ways.

Repurposing data yields new data that is worth *saving*. Even original data can be worth saving, just like some web pages are worth bookmarking to some users. Once there is support for saving data, there is a need to interact with the saved data in the long term. This involves *browsing* and *searching* through the data later, which might warrant *organizing* it earlier on.

Finally, an ultimate goal for data collecting and repurposing might be *publishing* it back into the Web.

Thus, a data-centric browser should support the following features:

- retrieve data from any web source;
- merge data from several sources;
- edit data efficiently;
- visualize data in many ways;
- browse/search through data;
- save data permanently;
- organize data;
- publish data expressively.

The various components in this thesis provide answers to how these features can be supported. More research is needed to fit them all together into a coherent whole and provide a seamless experience for moving data out of and into the Web.

6.2 Discussion

Now at the end of the thesis, let me venture outside the realm of computer science into the larger social, political, and economic contexts surrounding the idea of a data-centric Web.

6.2.1 Data Publishing as Empowerment

The diversity of interests on the Web brings big profits to online retailers. On Amazon, there are hundreds of thousands of unpopular books, each selling only a few to a few hundred copies, whose combined profit rivals that of the few hundred

books that sell millions of copies. This observation is explored in “The Long Tail” [40].

If we rank the various kinds of information that people want to publish on the Web by their popularity, we should also get a long tail (Figure 6.1): a few dozens of readily recalled kinds of information such as consumer products, news articles, events, locations, photos, videos, music, and software projects populate the massive head of the curve while hundreds of thousands of unpopular topics, such as sugar package collection and lock picking, spread thinly over the long tail. Clustering search engines such as clusty.com show dozens of sites of each unpopular topic. Wikipedia’s several thousand categories and DMoz’s half a million categories show how long the tail is.

Blogs have made publishing text articles even easier than writing HTML documents from scratch. Wikis let small to large communities collaboratively grow and maintain reasonably-sized web sites of text articles. Multimedia publishing services like Flickr and YouTube, complemented by the arrival of affordable consumer cameras, also add to the publishing power of the mass population. These publishing venues are now widely adopted and highly appreciated for giving the mass population text-based and multimedia publishing power rivaling large, professional, well-funded organizations. Wikipedia has grown large enough to be compared against Encyclopedia Britannica [49]. “Citizen journalism” [2] can deliver news faster and from more angles than news networks.

As blogs, wikis, and other publishing mechanisms spread, they become commonplace. A well-styled web site is now the baseline—the least to be expected. It sparks no surprise nor delight, and does not carry any authority just by its look. To set itself apart, a web site must differentiate against other sites by other factors, such as the expensive features that it can afford or the perceived objectivity in which it communicates its view. For example, to report a presidential election, a web site might show a map plotting per-state support for each political party, and a timeline coupled with the map to show how such support has changed over time. The site might even make accessible the data that fuels the map and the timeline so that

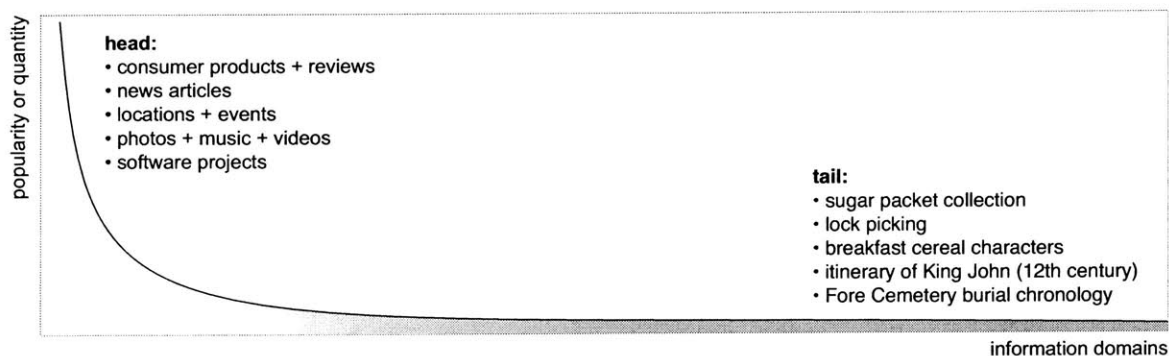


Figure 6.1. The Long Tail of Information Domains.

its visitors can double-check the validity of its visualizations. In doing so, the site builds its public image of being objective and open.

In my research, Sifter and Potluck, which allow casual users to repurpose and mix data from the Web, can be seen as anticipation of such a future of data-rich web sites. To complement, Exhibit gives the mass population ammunition to compete against large content providers as the two sides co-evolve toward that future.

6.2.2 The Data Market

The motivations for publishing data mentioned in the previous section are based on competition. Publishing data accessorizes one's web site and sets it apart from the rest. Reputation can be accumulated if one's data is reused for some grand purposes or by some reputable parties. If quality of data is a competitive advantage—a reasonable assumption—then competition among publishers will drive data quality upward. Perhaps economic theories can be applied to set the desirable conditions that drive this *data market* to produce the desired data in the end.

If there will indeed be a market in which data is traded for reputation or sold for money as small-time publishers try to differentiate among themselves, new industries will spring up to support their needs. Software and services for collecting, editing, managing, and hosting data are in demand. Data needs to be verified and certified by independent parties. Licenses are required to facilitate reuse. The right tools and services produced by these industries can help develop the market.

There is already some infrastructure for and evidence of such a market. For example, online consumer-oriented database and spreadsheet services like DabbleDB [5] let people collect and share data. Many Eyes [16] and Swivel [35] let people construct data visualizations. Sense.us [52] lets people collaboratively scrutinize data visualizations, pointing out anomalies and explaining trends. The Science Commons [29] are working on licenses for scientific data.

6.2.3 Data as Common Goods

Building such a data-rich Web has long been advocated by the Semantic Web project. Semantic Web researchers initially appealed to the vision in which software agents could harvest information on the Web and perform information-centric tasks on behalf of people. For that vision to work, the Semantic Web must be sufficiently complete and coherent such that software agents have adequate and reliable data to work on. This strategy requires people to believe in the vision and then to cooperate to build the Semantic Web for the benefits of all humanity in the future. As there is no immediate benefit but much initial labor required, this strategy mostly appeals to and brings together far-thinking, altruistic individuals. However, if the labor required is small enough and the vision very appealing, this strategy might even appeal to the mass population, as in many cases where *crowdsourcing* [3] does work. For example, there have been efforts by Google and OpenStreetMap to let web users edit street maps for countries in which street maps cannot be obtained

from the government. FreeBase [8] intends to let web users build a global database equivalent to Wikipedia.

We can also imagine smaller efforts in which geographically small communities collaborate to maintain databases of their resources for emergency purposes. Semantic MediaWiki [73] might be a good starting point toward this end. When a natural disaster strikes, it is desirable to get a big picture of the situation as quickly as possible. How many adults, children, elders, and physically disabled are in the disaster zone? How many vehicles are available? Does anyone own a boat if the disaster is a flood? Where do the doctors and nurses live? Where do mechanical engineers and electricians live? Which buildings are most likely still standing and capable of housing a certain number of people? Survivors can even help to collect data on the fly wherever they are. People outside the disaster zone can help organize and analyze the collected data.

Even before disasters strike, these community-maintained databases can be anonymized and pooled together to give a big picture of the whole country. Through incentive schemes, the population can be redistributed so that each region within the country can be as autonomous as possible, having all the skills and resources needed for disaster response.

6.2.4 Data as Culture

To engage the mass population for the purpose of building a data Web, usable technologies may not be enough. Various media machineries might need to be called upon to spread the meme of data appreciation to every corner of society.

Television shows such as “CSI: Crime Scene Investigation”, which show unrealistic scientific capabilities and romanticize crime scene evidence in rich visualizations, have raised real-world expectations of forensic science [4]. More students are enrolling in forensic science programs recently. A similar means can be used to increase the public’s expectation for the availability of data that helps during crises. Such expectation might in turn increase the public’s willingness to contribute to the community-maintained databases suggested previously.

The seed of appreciation for data can also be sown early in grade school education. Young children can be given tools and instructions for experimenting with data so to develop skills and tastes for data. Just as science kits are sold to pique children’s interests in science and give them hands-on experience with scientific experiments at an early age, “data kits” can also be made available to children.

What would it be like to live and grow up in a data-rich society? Would it be different from the *information*-rich society in which we are currently living? We are already facing the information overload problem. Would it not be worse to be overloaded with raw data? These questions remain in the realm of media studies, beyond the scope of this thesis.

6. CONCLUSION

7. BIBLIOGRAPHY

- [1] AdaptiveBlue, Browse Smarter. <http://adaptiveblue.com/>.
- [2] Citizen journalism. http://en.wikipedia.org/wiki/Citizen_journalism.
- [3] Crowdsourcing. <http://en.wikipedia.org/wiki/Crowdsourcing>.
- [4] CSI Effect. http://en.wikipedia.org/wiki/CSI_Effect.
- [5] DabbleDB. <http://dabbledb.com/>.
- [6] Dapper: The Data Mapper. <http://www.dapper.net/>.
- [7] DBpedia, Querying Wikipedia like a Database. <http://dbpedia.org/>.
- [8] Freebase. <http://freebase.com/>.
- [9] Gallery. <http://gallery.menalto.com/>.
- [10] Google Base. <http://base.google.com/>.
- [11] Google Maps. <http://maps.google.com/>.
- [12] Greasemonkey. <http://greasemonkey.mozdev.org/>.
- [13] Introducing JSON. <http://www.json.org/>.
- [14] Linked Data - Design Issues. <http://www.w3.org/DesignIssues/Linked-Data.html>.
- [15] Longwell. <http://simile.mit.edu/wiki/Longwell>.
- [16] Many Eyes. <http://services.alphaworks.ibm.com/manyeyes/home>.
- [17] microformats. <http://microformats.org/>.
- [18] mjt. <http://mjtemplate.org/>.
- [19] NetSnippets. <http://www.netsnippets.com/>.
- [20] Ning - Create your own Social Networks! <http://www.ning.com/>.
- [21] Online Spreadsheets - EditGrid. <http://www.editgrid.com/>.
- [22] Operator. <https://addons.mozilla.org/en-US/firefox/addon/4106>.
- [23] Protégé Ontology Editor and Knowledge Acquisition System, The. <http://protege.stanford.edu/>.
- [24] PubMed Home. <http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed>.

7. BIBLIOGRAPHY

- [25] Rdf in Html :: Embedded RDF Wiki :: Talis. <http://research.talis.com/2005/erdf/wiki/Main/RdfInHtml>.
- [26] RdfPath. <http://esw.w3.org/topic/RdfPath>.
- [27] RDFa Primer 1.0. <http://www.w3.org/TR/xhtml-rdfa-primer/>.
- [28] Resource Description Framework (RDF) / W3C Semantic Web Activity. <http://www.w3.org/RDF/>.
- [29] Science Commons. <http://sciencecommons.org/>.
- [30] Search-based User Interaction on the semantic web, a survey of existing systems. <http://media.cwi.nl/survey/>.
- [31] Semantic Web project. <http://www.w3.org/2001/sw/>.
- [32] Spotfire. <http://spotfire.com/>.
- [33] START, Natural Language Question Answering System. <http://start.csail.mit.edu/>.
- [34] Swoogle, semantic web search. <http://swoogle.umbc.edu/index.php>.
- [35] Tasty Data Goodies - Swivel. <http://www.swivel.com/>.
- [36] Topher's Breakfast Cereal Character Guide. <http://www.lavasurfer.com/cereal-guide.html>.
- [37] TrimQuery. <http://code.google.com/p/trimpath/wiki/TrimQuery>.
- [38] XML Path Language (XPath). <http://www.w3.org/TR/xpath>.
- [39] Allen, R.B. Retrieval from facet spaces. *Electronic Publishing*, vol. 8 (2&3), pp. 247–257, 1995.
- [40] Anderson, C. The Long Tail: Why the Future of Business is Selling Less of More. *New York: Hyperion, 2006*.
- [41] Barrett, R., P. Maglio, and D. Kelleym. How to personalize the web. *CHI 1997*.
- [42] Berners-Lee, T., J. Hendler, and O. Lassila. The Semantic Web. *Scientific American, May 2001*.
- [43] Berners-Lee, T., et. al. Tabulator: Exploring and Analyzing linked data on the Semantic Web. *SWUI 2006*.
- [44] Bizer, C., E. Pietriga, D. Karger, R. Lee. Fresnel: A Browser-Independent Presentation Vocabulary for RDF. *ISWC 2006*.
- [45] Bolin, M., M. Webber, P. Rha, T. Wilson, and R. Miller. Automation and Customization of Rendered Web Pages. *UIST 2005*.
- [46] Dontcheva, D., S. Drucker, G. Wade, D. Salesin, and M. Cohen. Summarizing Personal Web Browsing Sessions. *UIST 2006*.
- [47] English, J. M. Hearst, R. Sinha, K. Swearingen, and K.P. Yee. Flexible Search and Browsing using Faceted Metadata. Unpublished manuscript, 2002. Available at <http://flamenco.berkeley.edu/papers/flamenco02.pdf>.
- [48] Faaborg, A. and H. Lieberman. A Goal-Oriented Web Browser. *SIGCHI 2006*.
- [49] Giles, J. Internet encyclopaedias go head to head. *Nature, 2005*. <http://www.nature.com/news/2005/051212/full/438900a.html>.
- [50] Halevy, A.Y. (editor), N. Ashish, D. Bitton, M. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka. Enterprise Information Integration: Successes, Challenges and Controversies. *SIGMOD 2005*.

- [51] Hammer, J., H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge. The Stanford Data Warehousing Project. *IEEE Data Engineering Bulletin*, v.18, n.2, pp. 41-48, 1995.
- [52] Heer, J., F. Viégas, M. Wattenberg. Voyaers and Voyeurs: Supporting Asynchronous Collaborative Information Visualization. *SIGCHI 2007*.
- [53] Hildebrand, M., J. van Ossenbruggen, L. Hardman. /facet: A Browser for Heterogeneous Semantic Web Repositories. *ISWC 2006*.
- [54] Hogue, A. and D. Karger. Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web. *WWW 2005*.
- [55] Joachims, T., D. Freitag, and T. Mitchell. WebWatcher: a tour guide for the World Wide Web. *IJCAI 1997*.
- [56] Kalfoglou, Y. and M. Schorlemmer. Ontology Mapping: The State of the Art. *The Knowledge Engineering Review*, Volume 18, Issue 1, 2003.
- [57] Lerman, K., L. Getoor, S. Minton, and C. Knoblock. Using the structure of Web sites for automatic segmentation of tables. *SIGMOD 2004*.
- [58] McDowell, L., O. Etzioni, S. Gribble, A. Halevy, H. Levy, W. Pentney, D. Verma, and S. Vlasheva. Mangrove: Enticing Ordinary People onto the Semantic Web via Instant Gratification. *ISWC 2003*.
- [59] Miller, R. and B. Myers. Multiple Selections in Smart Text Editing. *IUI 2002*.
- [60] Oren, E., R. Delbru, S. Decker. Extending faceted navigation for RDF data. *ISWC 2006*.
- [61] Plaisant, C., B. Shneiderman, K. Doan, and T. Bruns. Interface and data architecture for query preview in networked information systems. *ACM Transactions on Information Systems*, 17(3):320-341, 1999.
- [62] Quan, D., D. Huynh, and D. Karger. Haystack: a platform for authoring end-user Semantic Web applications. *ISWC 2003*.
- [63] Reis, D.C., P.B. Golgher, A.S. Silva, and A.F. Laender. Automatic Web news extraction using tree edit distance. *WWW 2004*.
- [64] schraefel, m. c., D. Smith, A. Russel, A. Owens, C. Harris, and M. Wilson. The mSpace Classical Music Explorer: Improving Access to Classical Music for Real People. *MusicNetwork Open Workshop, Integration of Music in Multimedia Applications*, 2005.
- [65] 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. *WWW 2002*.
- [66] Shen, Y., and D. Karger. U-REST: An Unsupervised Record Extraction System. Poster, *WWW 2007*.
- [67] Shneiderman, B. Dynamic queries for visual information seeking. *IEEE Software*, 11:6, 70-77, 1994.
- [68] Sinha, V. and D. Karger. Magnet: Supporting Navigation in Semistructured Data Environments. *SIGMOD 2005*.
- [69] Suchanek, F.M., G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge - Unifying WordNet and Wikipedia. *WWW 2007*.

7. BIBLIOGRAPHY

- [70] Sugiura, A. and Y. Koseki. Internet scrapbook: automating Web browsing tasks by demonstration. *UIST 1998*.
- [71] Tai, K.-C. The tree-to-tree correction problem. *J. Association of Computing Machinery*, 26(3):422–433, July 1979.
- [72] Tufte, E. Visual Explanations: Images and Quantities, Evidence and Narrative. *Graphics Press*: Cheshire, Connecticut, 1997.
- [73] Völkel, M., M. Krötzsch, D. Vrandečić, H. Haller, R. Studer. Semantic Wikipedia. *WWW 2006*.
- [74] Wang, J.-Y., and F. Lochovsky. Data extraction and label assignment for Web databases. *WWW 2003*.
- [75] Wong, J. and J. Hong. Making Mashups with Marmite: Towards End-User Programming for the Web. *SIGCHI 2007*.
- [76] Yan, B., M. Frank, P. Szekely, R. Neches, and J. Lopez. WebScripter: Grass-roots Ontology Alignment via End-User Report Creation. *ISWC 2003*.
- [77] Yee, P., K. Swearingen, K. Li, and M. Hearst. Faceted Metadata for Image Search and Browsing. *SIGCHI 2003*.
- [78] Zhai, Y., and B. Liu. Web data extraction based on partial tree alignment. *WWW 2005*.