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.

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for browsing data collections on web sites by J. English, M. Hearst, R. Sinha, K. Swearinen, and K.P. Yee in an unpublished manuscript in 2002 [47].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

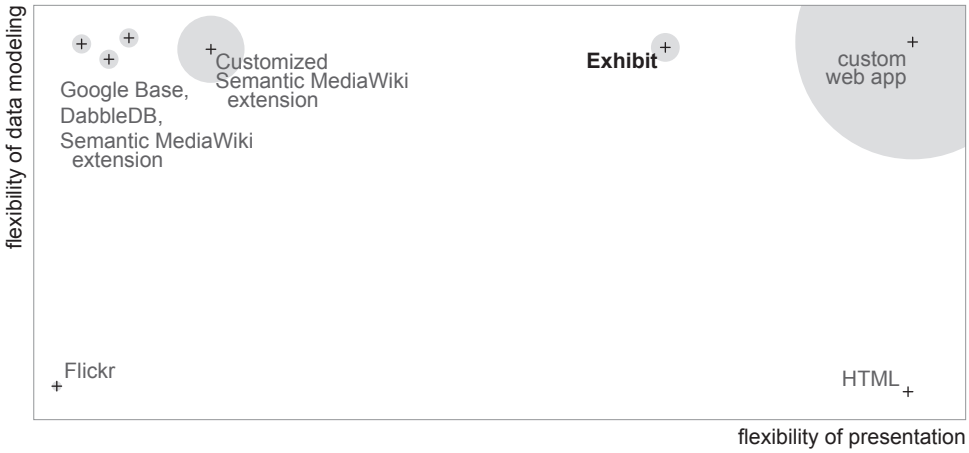

Figure 2.1. Flexibility of presentation and data modeling as well as the efforts required to adopt and use (circle size) for various publishing frameworks.

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

```
            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 XPaths. 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 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 pages 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 them 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 delegate 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.

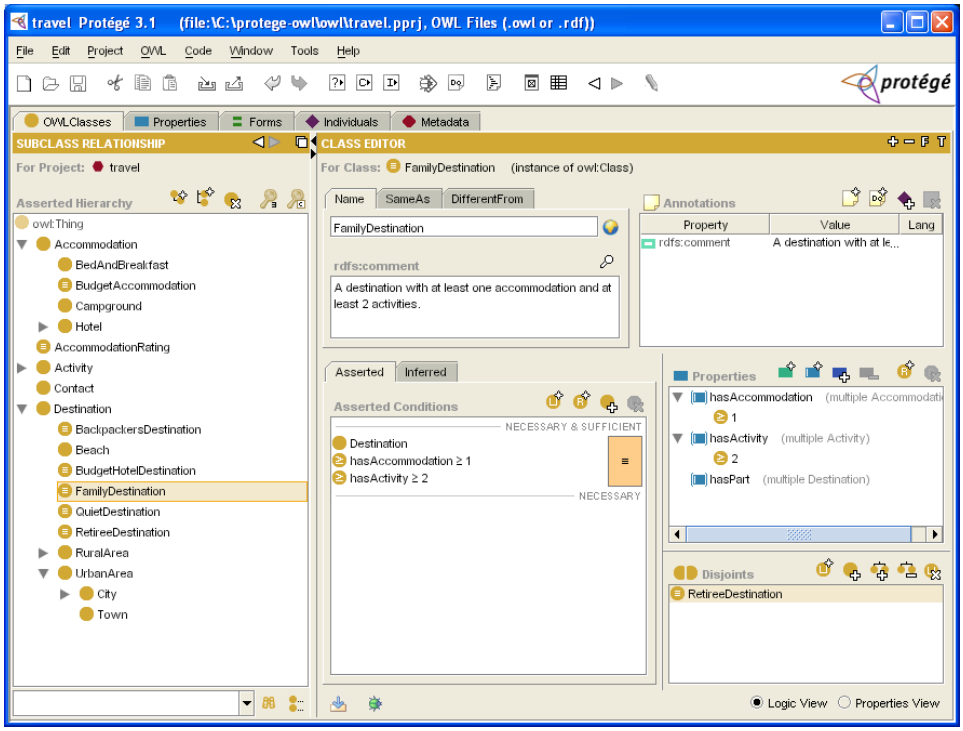

Figure 2.2. Professional ontology alignment tools, such as Protégé [23], are too advanced for casual users. They require understanding of abstract concepts like classes, class hierarchies, inheritance, inference, etc. (Image from http://protege.stanford.edu/.)

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.

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.