Standards opportunities around data-bearing Web pages

Citation: Karger, D. "Standards Opportunities Around Data-Bearing Web Pages." Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 371, no. 1987 (February 18, 2013): 20120381–20120381.

As Published: http://dx.doi.org/10.1098/rsta.2012.0381

Publisher: Royal Society, The

Persistent URL: http://hdl.handle.net/1721.1/87043

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Standards Opportunities around Data-Bearing Web Pages

David Karger

January 17, 2013

Abstract

The evolving web has seen ever-growing use of structured data, thanks to the way it enhances information authoring, querying, visualization, and sharing. To date, however, most structured data authoring and management tools have been oriented towards programmers and web developers. End users have been left behind, unable to leverage structured data for information management and communication as well as professionals.

In this article, I will argue that many of the benefits of structured data management can be provided to end users as well. I will describe an approach and tools that allow end users to define their own schemas (without knowing what a schema is), manage data, and author (not program) interactive web visualizations of that data using the web tools with which they are already familiar, such as plain web pages, blogs, wikis, and WYSIWYG document editors. I will describe our experience deploying these tools and some lessons relevant to their future evolution.

1 Introduction

The web dramatically simplified, and thus democratized, information authoring and sharing. Anyone whose organization was operating a web server could author content and make it available for all web users to access with a single click. Unlike the plain-text content of Usenet newsgroups, this content could be beautified by the user who managed format and layout and embedded images. Users could control not only their content but also how it was presented. The result was an explosion of end-user-authored content that became available to others throughout the world.

The web has evolved to make authoring even easier. Blogs allow users to post content for themselves and wikis help them to manage content collaboratively. WYSIWYG HTML editors abound, freeing users of the need to see, understand, or edit HTML source code. Even prior to such editors, in the early days of the web, many users managed to publish without learning HTML (or CSS) as they could simply copy someone else's document and modify small portions of the content.

But the continuing evolution of the web has left end users behind in an important way. Professional web sites make liberal use of *structured data*, which is often easier to author, manage, query, and reuse, and which supports powerful interactive visualization and manipulation interfaces. The tools typically used to support this work—databases, templating web engines, and programmatic visualization APIs—are beyond the reach of many end users. Such users remain limited to plain (however nicely formatted) text and images. They cannot manage their data or communicate it as effectively as professional sites.

In this article, I will argue that we can close this gap. By framing the problem properly, defining the right web standards, and giving users the right tools, we can enable end users to manage and publish structured data via already familiar workflows, without acquiring advanced programming skills. The results can rival the web interactions offered on professional sites. I will describe several prototype tools that we have created to pursue this objective, our experience deploying them, and some of the lessons we have learned and questions that have arisen from the experience. With a fully fleshed out set of these end-user data-oriented tools, it may be possible to spark a simplification and democratization of structured data authoring on the web that could imitate the web's original, text-based revolution.

Many articles on the Semantic Web tout its value of the web as a whole—the benefits of creating "open linked data" that fits standardized schemas and can be connected and aggregated in powerful ways. These benefits generally accrue to the *consumers* of the data. Often unaddressed are motivation (other than altruism) for *owners* and *authors* to publish this data. This article takes a complementary perspective, emphasizing the benefits that can accrue to authors *independent* of the behavior of other web sites. Such an approach can provide incentives that do not suffer from the "chicken and egg" problem faced by some Semantic Web efforts—where the benefits accrue only if all others adopt the same approach. By providing individual incentives, end-user authoring tools may help accelerate the adoption of structured data and a move towards the Semantic Web that will ultimately yield substantial social benefit.

2 The Power and Pain of Structured Data

In their effort to provide users with high-quality information interaction, many modern web sites exploit structured data. Precisely defining structured data would require a substantial investment of philosophical energy, so I'll work by example instead. Information in tables is structured, because you can refer to the contents of a particular cell at a certain row and column. Another structured data representation is *entity-relational models,* where each entity has a number of named *properties* that receive *values*. In contrast, I'll refer to text data as unstructured, since decomposing it into its intended entities and relations requires understanding of the words of the language itself and not just the format. A key requirement and benefit of structured data is that one can address and define operations on all instances of some part of the structure—sorting a spreadsheet's *rows* by the values in some *column*, or filtering *entities* according to the value of a given *property*. In contrast, given the complexity of grammar and pronouns, it is relatively difficult to select all sentences whose subject is a particular entity.

Structured data can dramatically simplify the process of maintaining a web site. In a web site of static pages, adding a new information item means laboriously copying and pasting web page elements to create new content that looks like the rest. If instead information objects are stored in a database, the web author can create a single *template* that describes how a given class of information objects should be presented, and rely on a *templating engine* that automatically fills the template with values from the properties of any item being presented. The author can then can bulk-update a database table using specialized forms or a spreadsheet editor, and leave it to the templating engine to show the modified information. Conversely, with a single modification to the template the author can update the way every single item is presented on the site. Even if the same information is presented in multiple locations, the author can update it once and know that the change will propagate; in contrast, each assertion of a given fact in an unstructured document must be separately and manually updated. Data updates can immediately show up in faceted browsing interfaces, without the author needing to rebuild static index pages.

Structured data also improves the experience of a user interacting with a site. For example, a user seeking cell-phones at a web site like CNET will find a list of candidate cell-phones that is highly structured, with attributes such as price, rating, size, carrier, weight, and presence or absence of features such as camera and keyboard. A user can sort on any of these features. He can also filter on combinations of them using the now-pervasive paradigm of *faceted browsing*—a list of values for each attribute is shown, and the user can select a set of values to filter the items to those having those values of the attribute. Faceted browsing cues users about ways they can explore the collection, instead of presenting them with a dauntingly blank text-search box and leaving it up to them to determine which searches actually yield results [YSLH03]. The presentation of all items in a consistent template makes it easier for the user to understand and scan the content (for example, by recognizing that the number in a specific region of the template is the item's price). Users can quickly assess the big picture of a large data set by examining *aggregate visualizations*, such as a price-performance *chart* showing tradeoffs of two key properties of microprocessors.

Much the same is true of most review or shopping sites, as well as others presenting large collections of information items such as libraries, scientific data repositories, museums, and directories.

It's clear that structured data is essential for these uses. One cannot offer faceted browsing unless the data has some properties to populate facets; templates place certain pieces/properties of the data in specific slots in the output; maps can only show things that have latitudes and longitude, and so on.

Unfortunately, it seems that many of these structured-data benefits can only be attained by the adoption of relatively sophisticated tools. Nearly every structured-data web site is backed by a database; someone needs to install and maintain that database, implement the database schema, and design the queries that produce the results that need to be shown. Someone has to program the pipeline that pulls data from the database and feeds it into the templating engine (where someone needs to program the templates). For rich visualizations, someone has to learn the API for the visualization tool and program the connection that feeds the data into it. All of these skills are beyond most users.

Thus, if users have structured data, they use ad-hoc solutions to manage it [VHAA11]. If they publish it to the web at all (often they do not) they are limited to plain tables [BMHK10], instead of the rich interactive visualizations offered by professional sites.

A notable demonstration of this phenomenon can be found in the world of journalism. Many journalists make heavy use of structured data to investigate their stories. But while these journalists will liberally quote anecdotes and aggregate summaries from these stories, they rarely incorporate the whole data set or a visualization of it. Instead, we have seen the emergence of a new class of *news app developers*—programmers in the news room who design and build one-off interactive visualizations for use with a single data set.

2.1 Approach

To address these problems, we draw insight from the design of HTML. This language recognized that the structure of a document could be described using a vocabulary of common elements—paragraphs, headings, italic or boldface sections, quotations, tables, lists, and so forth. HTML gives a syntax of *tags* for *describing*, not programming, the structure of a document using this standard vocabulary. This structured description can be used to automatically generate a layout of the document for visual presentation.

This article argues that there is a similar common vocabulary suited to the description of data visualizations and interactions, and that the HTML vocabulary can naturally incorporate these common visualization elements. Browsers (and other user agents) can then interpret these visualization descriptions to produce interactive visualizations, just as they currently interpret more basic document-structure tags to produce layouts of text documents.

I describe a particular prototype visualization vocabulary called Exhibit which has been deployed since 2007 and is currently in use on roughly 1800 web sites. Exhibit defines a data collection associated with a given HTML page (this data can be specified in table, or by linking to an external source such as a CSV (comma separated values) file or a Google spreadsheet). Exhibit offers *view* tags that specify aggregate visualizations of the data—such as maps, timelines, lists, and charts; *lens* template tags that specify the presentation of individual items; and *facet* tags that specify interactive filtering and search functionality. The Exhibit Javascript library interprets these tags to produce an interactive visualization of the page's data.

This declarative-visualization approach offers several benefits.

Declarative specification. HTML is a declarative language: the user describes *what* the layout should be instead of specifying the computation for creating the layout. A structured-data visualization language extension inherits the same benefits. It's far easier to indicate that a map should go in a certain place in a document by putting a map tag there, than it is to describe a computation that will initialize a map and place it in a specified place in the document. More people can author such "put this here" specifications than can write imperative programs in Javascript or other languages.

Pure client-side authoring. Because the view description is part of the page content, it can be handled entirely client side, as it is for example using our Exhibit library. Thus, nobody wishing to author a visualization has to worry about installation or management of server-side infrastructure. As with the regular web (now that web servers are pervasive) a user can simply author an HTML document and put it in a public location.

Page integration. An author can incorporate many distinct visualization tags on the same page, all referring to that page's data. She is in complete control of how the visualization elements interleave with the text and structural content of the page. She can place views side by side or at widely separate parts of the page. She can place facets in whatever relation she likes to the data views. Authors thus have full control of the presentation of their information narratives.

Low Startup Cost. Authors need learn almost nothing in order to create an Exhibit. A minimial exhibit can be created by adding three tags to an existing html document. Even better, authors can copy any HTML document containing a working exhibit and immediately have a working exhibit of their own. They can then begin incrementally adapting the exhibit to their own purposes, learning only as much about exhibit as they need to for their specific objectives.

Application independence. The use of an HTML vocabulary extension means that Exhibit descriptions can flow transparently through any application that handles HTML. As I will discuss below, this makes it easy to incorporate Exhibit editing as part of any HTML editor, or as part of a blogging framework.

Separating content and presentation. HTML recognized the importance of separating the description of a document's content from the decision about how to present the document. This meant that different tools could make their own decisions about optimizing the presentation of the given content. A mobile phone browser with a small screen might present the same data visualization differently from a desktop browser. A browser for the blind can use the structural description of a visualization to prepare an *audio* presentation of the content and leverage the structure to help its user navigate the content.

Longevity. Visualizations described using Exhibit can outlast specific implementations of Exhibit. The Exhibit vocabulary is quite small and well defined. Just as new browser implementations are frequently developed for interpreting the HTML vocabulary on new platforms, new Exhibit-vocabulary interpreters can be implemented at need. Contrast this with, for example, cloud-based visualizations that generally become useless when the cloud-service providing those visualizations shuts down.

3 Related Work

3.1 Declarative Visual Languages

Exhibit grew out of our previous work on declarative visual languages for structured data. The Haystack system [HKQ02] was a "semantic desktop" application that aimed to let end users manage their personal information using their own custom schemas and visualizations. Haystack used a declarative language for specifying how certain data should be laid out—for example, which properties of an email message should be shown, how those should be formatted, and how the set of all email messages should be arranged in a sorted list. These layouts declarations were specified by statements in the Resource Description Framework (RDF), an entity-relation model defined as part of the Semantic Web effort [Con04]. Haystack's visualization ontology included the concepts of views, lenses, and facets that later appeared in Exhibit. Haystack used the layout declarations to construct visualizations within its desktop application.

Following this work, we developed the Fresnel ontology [PBKL06]. Like Haystack's, Fresnel's visualization declarations are given using an RDF ontology. Fresnel benefitted from a more methodical consideration of what the right ontology should be, and careful attention to designing a declaration language that was device and application independent. Still, its core consisted of the same lens, view, and facet concepts that appeared in Haystack and earlier, for example in the OVAL system [MLF95]. Several different tools were built that understood the Fresnel vocabulary and used it to create visualizations. Longwell [PBKL06] used Fresnel specification to turn data into HTML documents, while IsaViz [Pie06] used the same declarations to present specialized network layouts of the data.

What differentiates Exhibit from these tools is our choice of HTML tags as the declaration language for visualizations. One can see this as a step backwards: the Exhibit language makes sense "only" to those applications that work with HTML. But this is its strength. HTML has become the pervasive language for presentation of information on the web. Far more tools and individuals can work with HTML than with more esoteric languages like RDF. Given that HTML is likely to be the target presentation language, it makes sense for visualizations also to be specified in HTML, eliminating the need for some extra process that transforms a different specification into HTML output.

3.2 Cloud Visualization Tools

An alternative to incorporating data and visualization as part of the web's HTML vocabulary is to offer desktop or cloud applications that build visualizations that can be embedded as "figures" in a web page. ManyEyes and Tableau are just two examples of this broad class. ManyEyes [DVWK08] (http://www.many-eyes.com/) is a web site where users can upload data and author visualizations of it, collaborate to discuss them or "clone" previous data and visualizations to create their modifications. ManyEyes offers diverse beautiful visualizations and has been applied to data from a wide variety of sources. Tableau (http://www.tableausoftware.com/) is one example of commercial "business intelligence" software. Tableau is a widely deployed commercial desktop application that lets users build a variety of data visualizations; these can then be exported to servers on the web. Both ManyEyes and Tableau are able to read a variety of structured data formats and offer simple yet powerful interactive interfaces for authoring visualizations.

Like many other platforms, ManyEyes and Tableau allow users to "embed" the data visualizations they create in their own web pages. Thus, to first order they meet the goal of letting end users publish data visualizations on the web. However, the embedding approach suffers from severe limitations.

The visualizations are not truly part of the containing web page—inside the embedded box, one can interact only with the visualization, while outside it, one can interact only with the containing page. This limits the ability of the author to interleave the layout of visualization and interaction elements with the rest of their page. Imagine a web where all images in a web page had to coexist in a single "image browser" frame that had to appear in one specific location on the page. This would obviously negatively impact the author's control of their presentation!

In a similar vein, from the perspective of the web page the embedded visualization is a single opaque region it cannot control. Thus, no styling of the containing page will propagate into the visualization. In the best case, the author will need to go through two separate styling processes to unify the styling of the containing page and the contained visualization. In the worst case, it will not be possible to style the two components consistently.

Another serious drawback of cloud visualization tools is that the author is often hostage to the continued existence of those tools. Many cloud services are high risk startups; they can disappear and take a user's visualizations with them.

3.3 Vertical Data Repositories

Although end-users rarely contribute generically structured data to the web, they are actively engaged in authoring structured data on a variety of domain specific sites that each manage data fitting a particular schema. Examples include restaurant reviews on Yelp, movie data on IMDB, book reviews on Amazon, recipes at Cooks.com, photo metadata on Flickr, and so on. Under this paradigm, the site owner chooses the schema, designs data input and visualization interfaces specific to the chosen schema, and installs and operates the database and templating engines. End users can author and the consume the data fitting the schema.

This approach is popular, well understood, and easy to implement. But it also comes with severe drawbacks. I argued above that the creation of such a content vertical is beyond the capabilities of end users. Thus, end users wishing to share content are at the mercy of the site developers. If the user wants a schema different from that chosen by the developer—for example, to record the "spiciness" of a recipe—they are out of luck. That information can go in comments, but cannot be used for visualization or faceted search. Similarly, it is impossible for the user to link data between distinct sites, associating "Greece" as a recipe origin with "Greece" on a travel site.

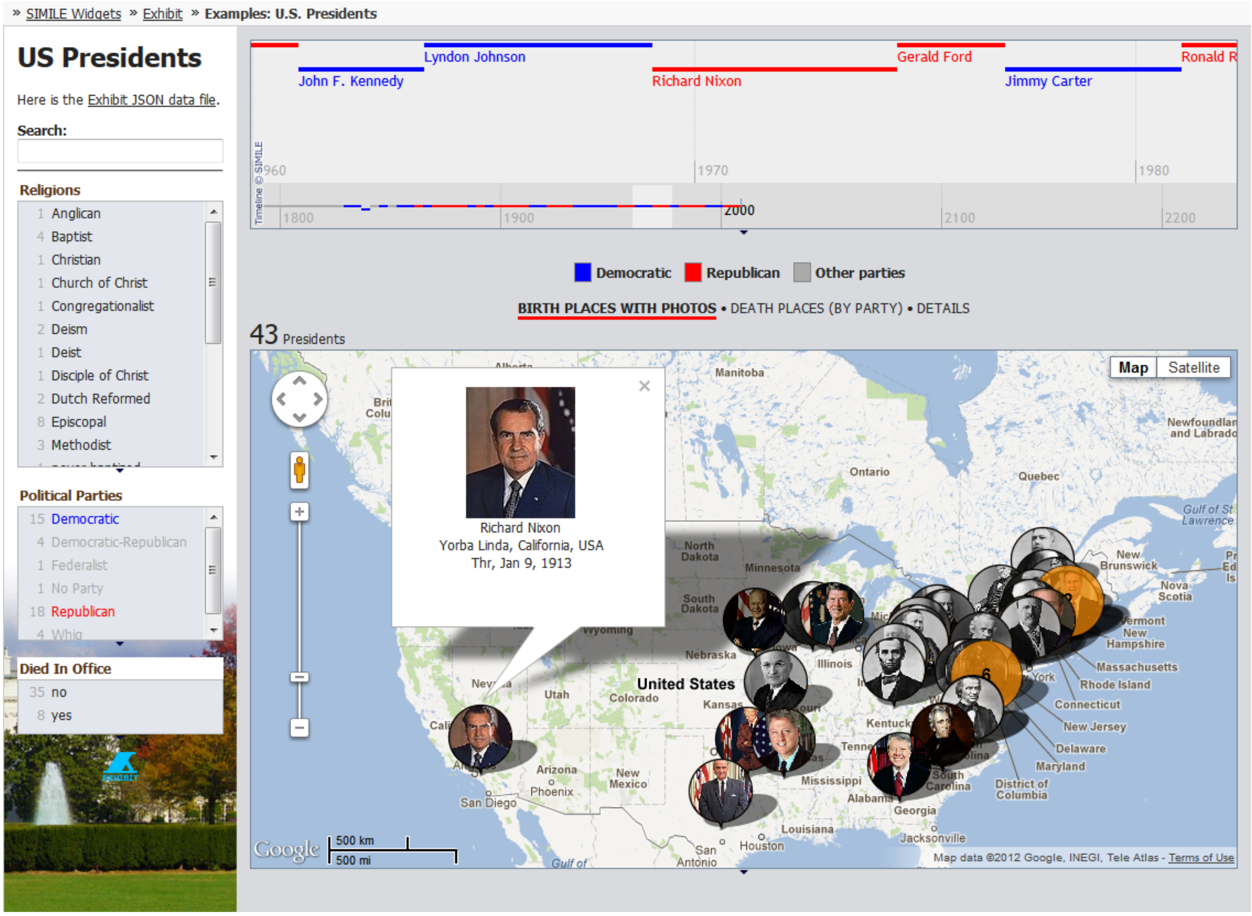
Even worse, vertical sites have only been created for relatively common data schemas. For a user with an unusual data set, there may be no appropriate site at all. This is a natural consequence of the fact that a developer is required to create these sites—developers are a scarce resource, and are deployed only when there is a sufficiently large user population.

4 Exhibit

Exhibit refers both to a particular prototype data-visualization extension to the HTML language, and to the Javascript library we created and deployed in 2007, which interprets tags from the visualization language and generates the specified visualizations. In this section I sketch the Exhibit language and library; more details can be found in our paper on Exhibit [HMK07]. A sample Exhibit can be seen in Figure 1; others can be seen in Figure 2. Once I have described Exhibit, I will describe how the first example is authored.

4.1 Data Model

Exhibit models data as a set of *items*, each of which has arbitrary *properties* that take on arbitrary *values*. For simplicity, Exhibit generally treats all values as text strings, although it interprets those strings as other things, such as numbers, dates, or URLs, when the context calls for it. This data can be presented



» SIMILE Widgets » Exhibit » Examples: U.S. Presidents

US Presidents

Here is the Exhibit JSON data file.

Search:

Religions

1	Anglican
4	Baptist
1	Christian
1	Church of Christ
1	Congregationalist
2	Deism
1	Deist
1	Disciple of Christ
2	Dutch Reformed
8	Episcopal
3	Methodist

Political Parties

15	Democratic
4	Democratic-Republican
1	Federalist
1	No Party
18	Republican
4	Whig

Died In Office

35	no
8	yes

🟦 Democratic 🟥 Republican ▨ Other parties

BIRTH PLACES WITH PHOTOS • DEATH PLACES (BY PARTY) • DETAILS

43 Presidents

Richard Nixon
Yorba Linda, California, USA
Thr, Jan 9, 1913

Figure 1: An exhibit of U.S. Presidents. A timeline view (color-coded by presidential party) is shown above a map of birthplaces. The left pane contains facets for filtering by religion, party, and whether the president died in office, as well as text search.

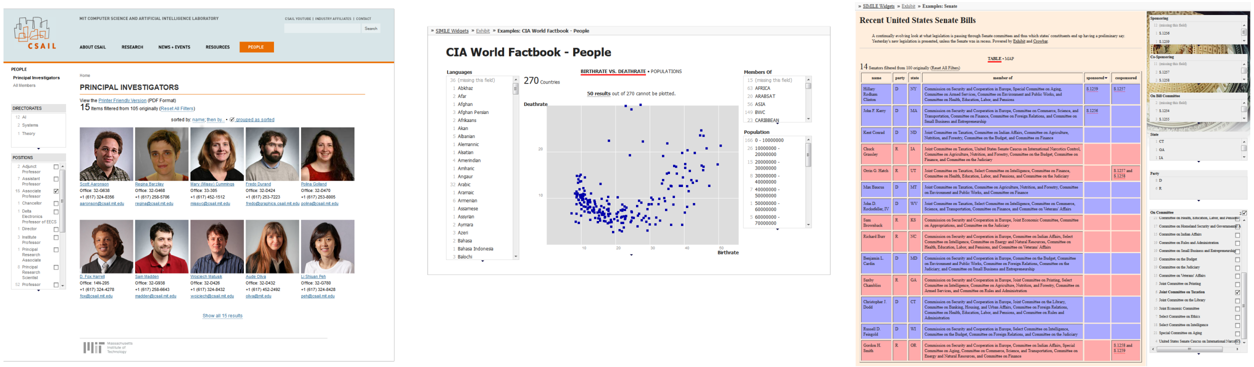


Figure 2: Other sample exhibits we created: a faculty directory using a thumbnail view, world population data presented using a scatter plot, and a set of senate bills presented in a tabular view

to Exhibit in a variety of formats: JSON, a comma- or tab-separated values file with one row per item, a Google spreadsheet, an html table, or various other formats.

To indicate the data he wishes to visualize, the author creates a reference to their data file by placing a standard HTML `link` tag in the document head, with the special additional attribute `rel="exhibit/data"` included to tell Exhibit that the given link points to a data file. Alternatively, the author can embed data in some formats, such as html tables, directly in the presentation document.

When authors create properties and values of items in their data files, they are implicitly defining schemas. Exhibit does not require that these schemas be defined explicitly or enforced—different items can have entirely different properties.

4.2 Visualization Elements

Exhibit defines visualization elements matching the categories already described: *views* of aggregate collections, *lenses* that template the presentation of individual items, and *facets* that filter the collection. Exhibit's interaction model is very straightforward: it presents, using the specified views, the collection of items that matches the current configuration of the facets. Individual items shown in the view are presented using the specified lenses.

Exhibit's *view* elements are described by adding an `ex:role="view"` attribute to an HTML tag such as a `div`. The author can specify a particular *type* of view using a `ex:viewclass` attribute; types of views include *Tile* (a list of items), *Thumbnail* (a set of thumbnail images), *map*, *timeline*, *scatterPlot* (where a dot on the plot is created for each item), *tabular* (a dynamic HTML table) and so on. A *viewPanel* can be used to multiplex several views; it provides tabs allowing the reader to select among the provided views.

Many of the views take additional attributes specifying their configuration. For example, the map view requires a `ex:latlng` attribute specifying which property of the data items should be looked at to find a latitude and longitude for plotting that item on a map, and allows an (optional) `ex:colorKey` attribute that specifies a property that the the map view should use to color code the items presented on the map. The tile view allows an `ex:orders` attribute specifying which property should be used as a "sort key" for ordering the list of items shown, while the tabular view takes an `ex:columns` property containing a comma-separated list of the properties of each item that should be shown in distinct columns of the tabular view. In all attributes, properties are specified by using the name of the given property in the data file—the column headings from a tabular file, the field names from a JSON file, and so on.

Facets to filter the data items are specified similarly—using an `ex:role="facet"` attribute on an HTML tag. Generally, facets filter on a particular property of the data items, which is specified using an `ex:expression` attribute that names a property as was done for views. When multiple facets are set by the reader, the collection is filtered to items that match *all* the facets—a conjunction of the constraints. The particular type of facet to use is indicated using an `ex:facetClass` attribute. Possibilities include the (default) `List` facet, which shows a list of values of the property. When the reader selects some of the values in the list, the collection of items is filtered down to items whose values for the property match the selected one. Other facet types include the `numericRange` or `slider` facets for filtering items whose property value is a number within the reader's specified range and a `textSearch` facet that filters to items whose text content matches the text query typed in the facet.

Lens templates are fragments of HTML indicating how an individual item should be displayed. For the most part they consist of standard HTML; however, any tag can be given an `ex:content` attribute that names a property. To display an item, Exhibit uses the lens but replaces each `ex:content` tag with the value of the specified property on the item being displayed. Lenses are used to render each item in a list or a thumbnail view; they are also used to render individual item "popups" when a reader clicks on a marker on the map, timeline, or scatter plot views.

4.3 Worked Example: U.S. Presidents

I now sketch how the elements we've just described can be assembled to produce the U.S. Presidents exhibit in Figure 1. As we've emphasized above, Exhibit uses an extension to the usual HTML vocabulary. Thus, the Presidents exhibit consists of a standard HTML document (plus CSS), augmented with some of Exhibit's particular tags. As a first step, the author creates a file containing the presidents' data. A portion can be seen in Figure 4.3. In this case, the file is a JSON file; however, it could equally be a google spreadsheet or a comma-separated-values document. Even without knowing the JSON format, the basic concept should be clear. Each president has a collection of properties (such as inDate and birthLatLng) and values for each property. In a spreadsheet or table, there would be a row for each item and a column for each property, with the value for a given item's property stored in the cell at the corresponding row and column.

Once the data file has been created, the author links it to the HTML file by adding a data link tag in the head of the document: `<link rel="exhibit/data" href="presidents.json">`. She also adds a link to the Exhibit Javascript library `<script src="exhibit-api.js">`. The page is now prepared to display widgets described by Exhibit vocabulary tags. For example, the "Religions" facet on the upper left is specified by the tag `<div ex:role="facet" ex:expression=".religion">`. Note that "religion" is one of the properties listed in the data file; the use of the ".religion" value for the `ex:expr` attribute in the facet tells exhibit which property the facet should filter on. Similarly, the text search facet above is specified using `<div ex:role="facet" ex:facetClass="TextSearch">`. The `TextSearch` value specifies that instead of the default "list of value" facet, the author wants a box where a text query can be typed; no expression is specified because the text search should apply to the full content of the items. The timeline at the top is created much the same way, by inserting a tag at the top of the HTML: `<div ex:role="view" ex:viewClass="TimeLine" ex:start=".inDate" ex:end=".outDate" ex:colorKey=".party">`. Again, the attributes of the tag specify properties from the data file that should be used to determine the start time (inDate), end time (outDate) and color (party) of the interval plotted for each item on the timeline.

Below the timeline, the author specifies a *view panel* by placing a `<div ex:role="viewPanel">`) tag. The view panel allows a reader to switch between several different views occupying the same display space. Multiple view tags can be placed inside the view panel tag; when instantiated, the view

```
{
 "items":[
  {"label":"George Washington",
   "type":"President",
   "imageURL":"http://upload.wikimedia.org/wikipedia/commons/thumb/8/88/Portrait_of_George_Washington.jpeg/100px-Portrait_of_George_Washington.jpeg",
   "url":"http://en.wikipedia.org/wiki/George_Washington",
   "presidency":"1",
   "term":["1","2"],
   "birth":"1732-02-22",
   "death":"1799-12-14",
   "inDate":"1789-04-30",
   "outDate":"1797-03-04",
   "index":"1",
   "dieInOffice":"no",
   "party":"No Party",
   "birthPlace":"Westmoreland County, Virginia, USA",
   "deathPlace":"Mount Vernon, Virginia",
   "religion":["Anglican","Episcopal","Deist"],
   "birthLatLng":"37.585241,-77.497343",
   "deathLatLng":"38.707778,-77.086389",
  },
  {"label":"John Adams",
   "type":"President",
   "imageURL":"http://upload.wikimedia.org/wikipedia/commons/thumb/3/3d/Gilbert_Stuart_John_Adams.jpg/101px-Gilbert_Stuart_John_Adams.jpg",
   "url":"http://en.wikipedia.org/wiki/John_Adams",
   "presidency":"2",
   "term":"3",
   "birth":"1735-10-30",
   "death":"1826-07-04",
   "inDate":"1797-03-04",
   "outDate":"1801-03-04",
   "index":"2",
   "dieInOffice":"no",
   "party":"Federalist",
   "birthPlace":"Braintree, Massachusetts, USA",
   "deathPlace":"Quincy, Massachusetts",
   "religion":"Unitarian",
   "birthLatLng":"42.222222,-71",
   "deathLatLng":"42.252778,-71.002778",
...
```

Figure 3: U.S. Presidents data

panel renders tabs that allow the reader to switch among the multiple views.

In this case, the first view placed inside the view panel, and the one display in Figure 1, is a map, specified using `<div ex:role="view" ex:viewClass="Map" ex:latlng=".birthLatLng" ex:iconURL=".imgURL">`. Also inside the view panel (but not displayed in the figure as they are not selected) are another map view and a *tabular view* showing a table of the items with columns selected from the available properties: `<div ex:role="view" ex:viewClass="Tabular" ex:columns=".term,.label,.party,.birth,.death">`.

The final step is to specify the lens template for individual items that appears when the reader clicks on a particular map icon or timeline interval. This template is a fragment of standard HTML with specific properties filled in. The lens for the map view is specified by placing the following HTML inside the map view tag:

```
<div class="map-lens" ex:role="lens">
 <div><img ex:src-content=".imageURL" /></div>
 <div ex:content=".label"></div>
 <div ex:content=".birthPlace"></div>
 <div ex:content=".birth"></div>
</div>
```

This fragment specifies which properties go where; in this case an `img` tag, whose `src` attribute is filled with the imageURL property, is placed above a line of text containing the label (name) of the president, which is above a line containing the birthplace, and another containing the birth year. Each of these lines refers to a property that can be found in the data file.

For simplicity I have neglected a number of additional parameters from the exhibit tags, such as parameters specifying heights and widths of various widgets, and have slightly simplified the data file

(I will discuss this simplification in Section 6), but this example provides enough detail to construct an Exhibit generally similar to that found in Figure 1

The process we've just described requires that an author directly edit HTML source code; certainly easier than propgramming, but a skill beyond many users. However, since the overall process is simply to add some HTML tags to a page, it should be clear that this kind of authoring could also be carried out by any WYSIWYG HTML editing tool that has been extended to handle Exhibit's special tags; I will describe an example of such a tool in Section 5. Equally important, new users need not start from scratch: they can copy any exhibit they find on the web and make small modifications to meet their needs, without a full understanding of the vocabulary. We'll elaborate on the approach in Section 6.3.

4.4 Implementation and Performance

Exhibit is implemented as a pure Javascript library that executes entirely on the client. On current browsers, it generally offers sub-quarter-second (perceived as "instantaneous") interaction response on exhibits of hundreds or even a thousand items. As the database grows (some exhibits have been created with as many as 27,000 items) Exhibit continues to function correctly but slows down linearly.

There are clearly many data sets that fit comfortably within these Exhibit parameters. So many that it would be foolish to sacrifice ease of use, flexibility or visualization power in search of performance. At the same time, there will always be data sets just slightly beyond Exhibit's comfortable reach, so it is worth understanding opportunities to improve Exhibit's scalability.

Three main bottlenecks impact Exhibit's performance and responsiveness. One is data loading: fetching data from the server and parsing it at the client can introduce a noticeable hesitation. This might be significantly improved if we explored "compressed" representations of the exhibit data. Standard compression could reduce the network transmission cost. Better representations that replace duplicate strings (there tend to be many, among items that share facet values) with references to a single copy could also speed parsing. Both compressions would require more work of the author to "prepare" their data for access.

Before much work is invested here, it is worth noting that, according to the HTTP Archive (http://www.httparchive.org/) the average number of bytes fetched by a browser when visiting a page from one of the top 1000 web sites in November 2012 exceeded one megabyte (including HTML, CSS, images, and script files). Meanwhile, the U.S. Presidents exhibit in Figure 1 uses a data file describing 44 presidents in 41 kilobytes, or less than one kilobyte per president. Clearly, even quite large data sets are unlikely to add significantly to the bandwidth involved in transmitting a single page. Indeed, Exhibit offers significant bandwidth optimization, as "pure" data uses far fewer bytes than "rendered" data. When server-side solutions present data, they run it through a templating engine that wraps each data item in substantial quantities of presentational HTML, which must then be transmitted. In contrast, Exhibit generates that presentational HTML on the client, eliminating the cost of transmitting it from the server.

A second performance bottleneck is database activity. Exhibit runs on its own, pure Javascript, non-optimized database that traverses hash tables to look up properties and values. There are obvious opportunities to improve this. Datavore [KPP$^+$12] is a pure Javascript database that uses a packed table representation and is able to support interactive-time querying of a million data rows. Looking further ahead, there has already been significant experimentation in incorporating a SQL database, and exposing a standard API to it, as part of the browser (cf. Mozilla/MySQL). While that effort has been temporarily derailed by arguments among browser vendors, I believe that it will ultimately bear fruit. Exhibit is a perfect candidate client of such a native browser database.

In a different approach, recent work on Exhibit version 3 has included *Backstage*, a server-side support engine for Exhibit. Backstage is designed to load data on the server side (saving exhibit the work of transmission and parsing). Exhibit then issues necessary queries to backstage, which processes them in native code on the server.

The final performance bottleneck for Exhibit, which database improvements cannot address, is visualization rendering. If an exhibit has 27,000 items, then rendering a list of them is going to involve a substantial amount of DOM manipulation which can be costly. Exhibit ameliorates this particular "list" problem by giving authors the opportunity to specify that only 10 items at a time should be shown, with pagination available to access the rest. But this only covers a single instance of the problem; the same problem arises in a facet that needs to display a large number of possible filtering values. A more general "frame buffer" approach can be applied to this case, of only rendering the portion of the DOM that is actually in view at a given time; this approach was explored by Nusser et al. [NCW$^+$07] who demonstrated its use in a Javascript client offering interactive web browsing and scrolling of tens of thousands of email messages.

The rendering problem extends beyond lists: placing tens of thousands of items on a map, scatter plot, or timeline is costly using today's APIs. It also extends to native visualization tools, where various approach have been tried such as not rendering until the data is filtered to a smaller set or rendering only sampled/summary information. These approaches could be applied in Exhibit as well.

As is usual in performance engineering, addressing just one of the three bottlenecks would be useless—all three need to be tackled to improve scalability. For example, in early testing of Exhibit's performance, it was determined that execution time was split almost exactly between database querying and visualization (DOM) rendering.

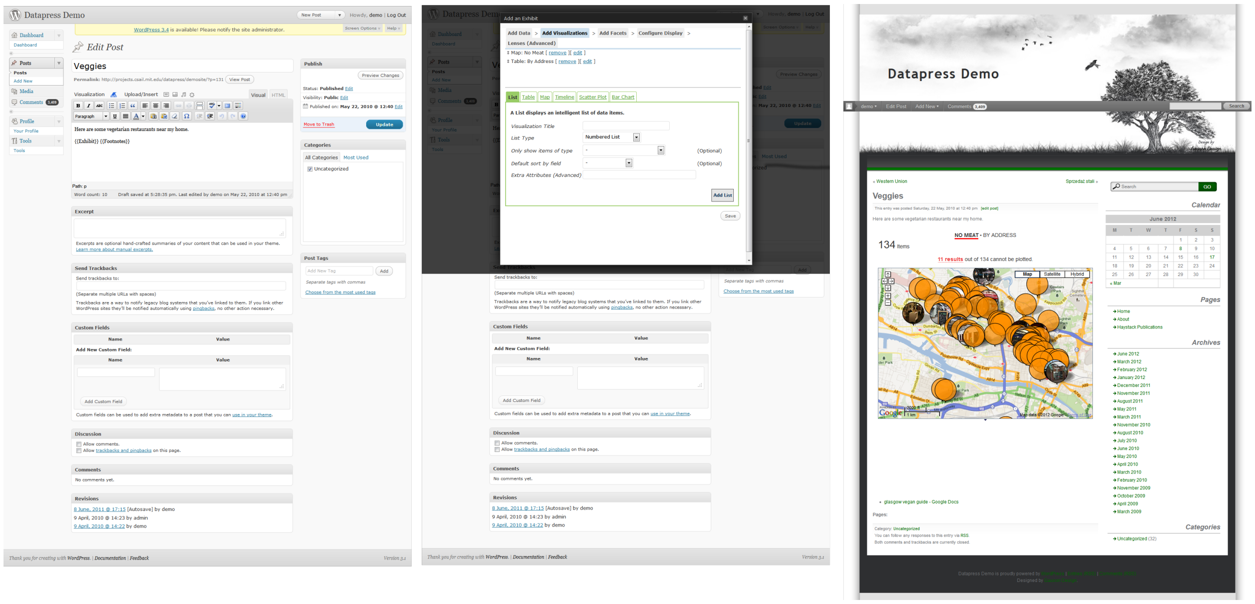
5 Extensions

Exhibit's thesis is that interactive-data authoring can and should be very similar to plain text and HTML authoring. It is therefore natural to consider adding Exhibit functionality to any environment where HTML is authored. We have created three demonstrations of this idea: Datapress [BMHK10], which incorporates Exhibit as a Wordpress extension, Wibit (also [BMHK10], which incorporates Exhibit as a Mediawiki extension, and Dido [KOL09], which combines Exhibit with a stand-alone WYSIWYG HTML editor.

Thanks to Exhibit's declarative HTML-based language, incorporating Exhibit in these tools is straightforward: the tools already work with HTML, so all that is necessary is to ensure that they generate or pass through Exhibit's scripts and special HTML tags; the application need not know anything about how those tags will actually be used for visualization. No communication between the application and the Exhibit library is required.

5.1 Datapress

Wordpress is a popular blogging platform which is used heavily for every imaginable type of text content. A small study [BMHK10] showed that data was also appearing on many Wordpress blogs, but almost always as static HTML tables or descriptive text. Rich visualizations could not be found. We incorporated Exhibit to provide those visualizations. Wordpress already comes with a WYSIWYG editing environment (tinyMCE), so we incorporated Exhibit editing as part of that editor. We used a wizard approach, providing a button that, when clicked, takes a user through a series of dialogs where she specifies the Exhibit she wants. The first dialog asks her to upload a suitable data file to her blog. Subsequent dialogs let her specify views, facets, and lenses. These dialogs use drop-down menus populated with property names extracted from the uploaded data file; the user specifies the entire visualization using point and click, without ever seeing the HTML that describes the Exhibit. After the user saves her page,



WordPress 3.4 is available! Please notify the site administrator.

Edit Post

Veggies

Permalink: http://projects.csail.mit.edu/datapress/demosite/?p=131 View Post

Here are some vegetarian restaurants near my home.

{{Exhibit}} {{Footnotes}}

Path: p

Word count: 10 Draft saved at 5:28:35 pm. Last edited by demo on May 22, 2010 at 12:40 pm

Status: Published Edit

Visibility: Public Edit

Published on: May 22, 2010 @ 12:40 Edit

☑ Uncategorized

Excerpt

Excerpts are optional hand-crafted summaries of your content that can be used in your theme. Learn more about manual excerpts.

Send Trackbacks

Send trackbacks to:

(Separate multiple URLs with spaces)

Trackbacks are a way to notify legacy blog systems that you've linked to them. If you link other WordPress sites they'll be notified automatically using pingbacks, no other action necessary.

Separate tags with commas

Choose from the most used tags

Custom Fields

Name	Value

Add New Custom Field:

Name	Value

Map: No Meat [remove][edit]

Table: By Address [remove][edit]

A List displays an intelligent list of data items.

Visualization Title

List Type: Numbered List

Only show items of type: - (Optional)

Default sort by field: - (Optional)

Extra Attributes (Advanced)

« Western Union

Sprzedaż stali »

Veggies

This entry was posted Saturday, 22 May, 2010 at 12:40 pm [edit post]

Here are some vegetarian restaurants near my home.

NO MEAT • BY ADDRESS

134 Items

11 results out of 134 cannot be plotted.

- glasgow vegan guide - Google Docs

Pages:

Category: Uncategorized

You can follow any responses to this entry via RSS.

Both comments and trackbacks are currently closed.

June 2012

M	T	W	T	F	S	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	
« Mar						

- Home
- About
- Haystack Publications

- June 2012
- March 2012
- February 2012
- January 2012
- December 2011
- November 2011
- August 2011
- May 2011
- March 2011
- November 2010
- August 2010
- July 2010
- June 2010
- May 2010
- April 2010
- March 2010
- February 2010
- November 2009
- October 2009
- April 2009
- March 2009

- Uncategorized (32)

Datapress Demo is proudly powered by WordPress | Entries (RSS) | Comments (RSS)

Designed by Gauson Design.

Figure 4: Using the Datapress wizard to author an Exhibit in Wordpress. Left: the Wordpress editor with additional "visualization" button. Middle: the "add a view" wizard dialog. Right: the final exhibit in a blog post.

the Datapress extension generates the HTML that describes the specified Exhibit. Datapress is shown in Figure 4

5.2 Wibit

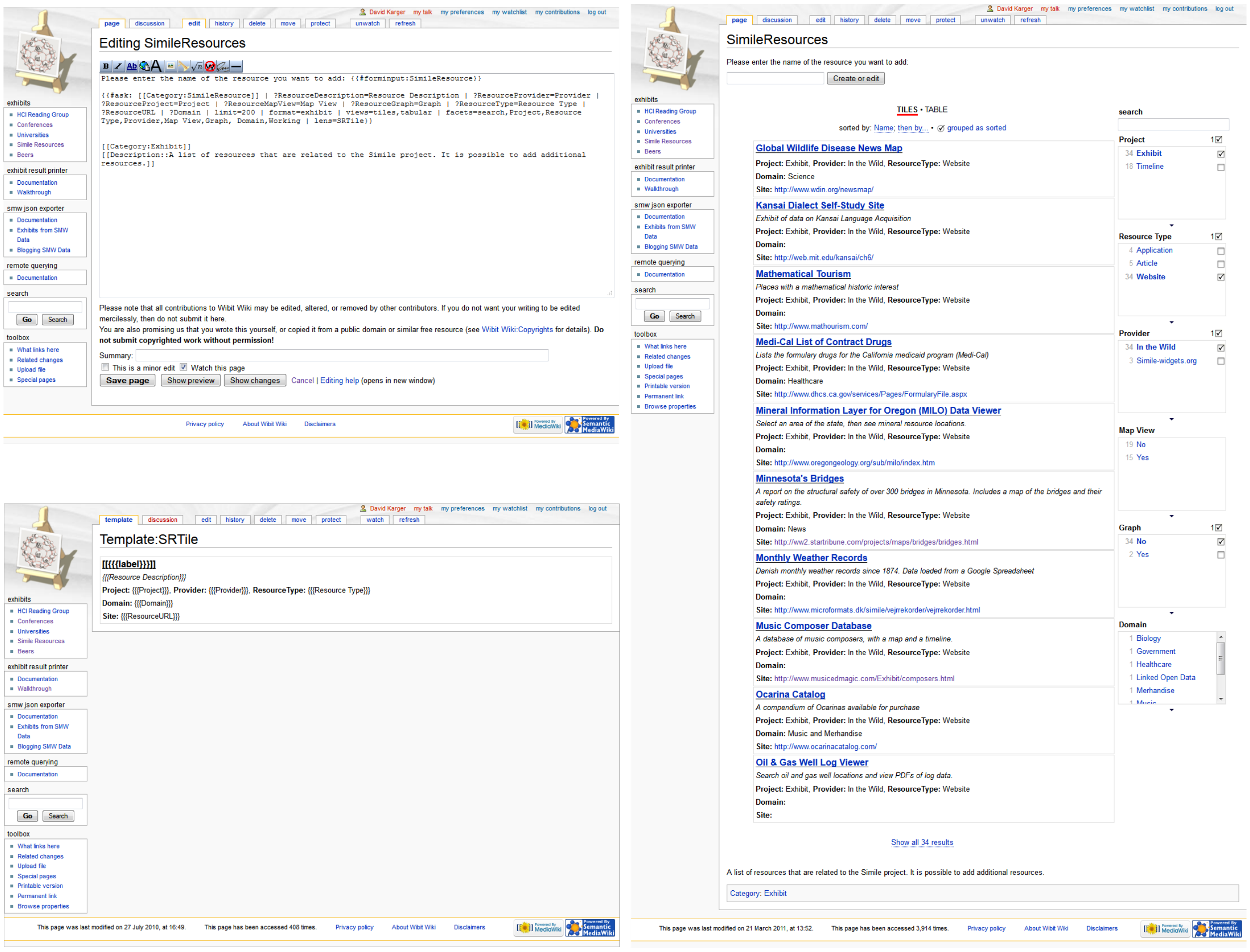
Like blogs for individual authoring, wikis have been broadly adopted for *collaborative* text authoring. So it is natural to extend them to support collaborative authoring of data visualizations. We created Wibit, an extension to the popular MediaWiki platform (which backs Wikipedia). Wibit is shown in Figure 5.

Rather then having users directly author HTML, either in source form or using a WYSIWYG editor, many wikis have chosen *Wikitext* as their canonical authoring language. This is an alternate syntax to HTML (one involving far fewer keystrokes, but plenty of esoteric special-character sequences) that describes document structure (lists, paragraphs, and tables) and styling (fonts, italics and underlining, etc.) To fit it in appropriately, we designed an Exhibit-specific extension to Wikitext for specifying Exhibit views and lenses, and leveraged the existing Wikitext template language for specifying lenses.

Exhibits are designed to present data, and data is not a basic concept of wikis. However, a powerful Mediawiki extension called Semantic Mediawiki introduces data to the system. With Semantic MediaWiki, structured data entered in "infoboxes" on wiki pages is parsed and stored into databases where it can be queried. With the Wibit and Semantic Mediawiki extensions in place, authors can create Exhibit visualizations by authoring a page that contains a Semantic Mediawiki data query together with an Exhibit visualization description that will present the data returned by the query.

5.3 Dido

Carrying the notion that visualization authoring is like text authoring to its logical extreme, Dido [KOL09] is a stand-alone HTML document that incorporates Exhibit as well as its own built-in WYSIWYG editor (see Figure 6. A user opens a Dido document in any web browser and can then interact with it like any



Editing SimileResources

```
Please enter the name of the resource you want to add: {{#forminput:SimileResource}}

{{#ask: [[Category:SimileResource]] | ?ResourceDescription=Resource Description | ?ResourceProvider=Provider |
?ResourceProject=Project | ?ResourceMapView=Map View | ?ResourceGraph=Graph | ?ResourceType=Resource Type |
?ResourceURL | ?Domain | limit=200 | format=exhibit | views=tiles,tabular | facets=search,Project,Resource
Type,Provider,Map View,Graph, Domain,Working | lens=SRTile}}


[[Category:Exhibit]]
[[Description::A list of resources that are related to the Simile project. It is possible to add additional
resources.]]
```

Please note that all contributions to Wibit Wiki may be edited, altered, or removed by other contributors. If you do not want your writing to be edited mercilessly, then do not submit it here.
You are also promising us that you wrote this yourself, or copied it from a public domain or similar free resource (see Wibit Wiki:Copyrights for details). **Do not submit copyrighted work without permission!**

Summary:
□ This is a minor edit ☑ Watch this page
Save page | Show preview | Show changes | Cancel | Editing help (opens in new window)

Template:SRTile

[[{{{label}}}]]
{{{Resource Description}}}
Project: {{{Project}}}, **Provider:** {{{Provider}}}, **ResourceType:** {{{Resource Type}}}
Domain: {{{Domain}}}
Site: {{{ResourceURL}}}

This page was last modified on 27 July 2010, at 16:49. This page has been accessed 408 times.

SimileResources

Please enter the name of the resource you want to add:
Create or edit

TILES • TABLE

sorted by: Name; then by... • grouped as sorted

Global Wildlife Disease News Map
Project: Exhibit, **Provider:** In the Wild, **ResourceType:** Website
Domain: Science
Site: http://www.wdin.org/newsmap/

Kansai Dialect Self-Study Site
Exhibit of data on Kansai Language Acquisition
Project: Exhibit, **Provider:** In the Wild, **ResourceType:** Website
Domain:
Site: http://web.mit.edu/kansai/ch6/

Mathematical Tourism
Places with a mathematical historic interest
Project: Exhibit, **Provider:** In the Wild, **ResourceType:** Website
Domain:
Site: http://www.mathourism.com/

Medi-Cal List of Contract Drugs
Lists the formulary drugs for the California medicaid program (Medi-Cal)
Project: Exhibit, **Provider:** In the Wild, **ResourceType:** Website
Domain: Healthcare
Site: http://www.dhcs.ca.gov/services/Pages/FormularyFile.aspx

Mineral Information Layer for Oregon (MILO) Data Viewer
Select an area of the state, then see mineral resource locations.
Project: Exhibit, **Provider:** In the Wild, **ResourceType:** Website
Domain:
Site: http://www.oregongeology.org/sub/milo/index.htm

Minnesota's Bridges
A report on the structural safety of over 300 bridges in Minnesota. Includes a map of the bridges and their safety ratings.
Project: Exhibit, **Provider:** In the Wild, **ResourceType:** Website
Domain: News
Site: http://ww2.startribune.com/projects/maps/bridges/bridges.html

Monthly Weather Records
Danish monthly weather records since 1874. Data loaded from a Google Spreadsheet
Project: Exhibit, **Provider:** In the Wild, **ResourceType:** Website
Domain:
Site: http://www.microformats.dk/simile/vejrrekorder/vejrrekorder.html

Music Composer Database
A database of music composers, with a map and a timeline.
Project: Exhibit, **Provider:** In the Wild, **ResourceType:** Website
Domain:
Site: http://www.musicedmagic.com/Exhibit/composers.html

Ocarina Catalog
A compendium of Ocarinas available for purchase
Project: Exhibit, **Provider:** In the Wild, **ResourceType:** Website
Domain: Music and Merhandise
Site: http://www.ocarinacatalog.com/

Oil & Gas Well Log Viewer
Search oil and gas well locations and view PDFs of log data.
Project: Exhibit, **Provider:** In the Wild, **ResourceType:** Website
Domain:
Site:

Show all 34 results

search

Project 1☑
34 Exhibit ☑
18 Timeline □

Resource Type 1☑
4 Application □
5 Article □
34 Website ☑

Provider 1☑
34 In the Wild ☑
3 Simile-widgets.org □

Map View
19 No
15 Yes

Graph 1☑
34 No ☑
2 Yes □

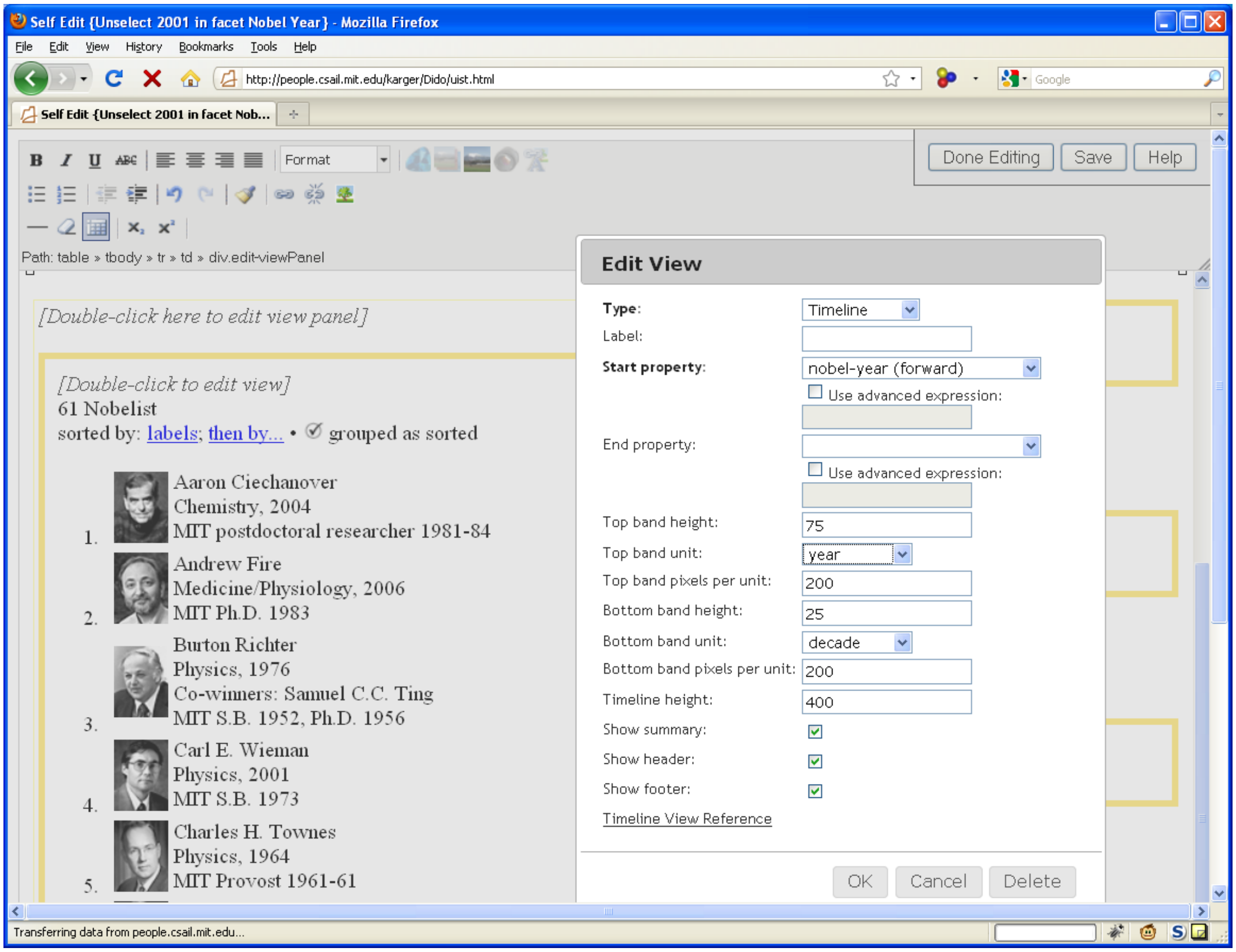
Domain
1 Biology
1 Government
1 Healthcare
1 Linked Open Data
1 Merhandise
1 Music

A list of resources that are related to the Simile project. It is possible to add additional resources.

Category: Exhibit

This page was last modified on 21 March 2011, at 13:52. This page has been accessed 3,914 times.

Figure 5: (top left) Wikitext is used to describe a query in a Semantic Mediawiki, and (bottom left) to describe a template (lens) for the individual results. (right) The desired exhibit of the results.


Self Edit {Unselect 2001 in facet Nobel Year} - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://people.csail.mit.edu/karger/Dido/uist.html
Google
Self Edit {Unselect 2001 in facet Nob...
Format
Done Editing
Save
Help
Path: table » tbody » tr » td » div.edit-viewPanel
OK
Cancel
Delete
Transferring data from people.csail.mit.edu...

[Double-click here to edit view panel]

[Double-click to edit view]
61 Nobelist
sorted by: labels; then by... • ☑ grouped as sorted

1. Aaron Ciechanover
Chemistry, 2004
MIT postdoctoral researcher 1981-84

2. Andrew Fire
Medicine/Physiology, 2006
MIT Ph.D. 1983

3. Burton Richter
Physics, 1976
Co-winners: Samuel C.C. Ting
MIT S.B. 1952, Ph.D. 1956

4. Carl E. Wieman
Physics, 2001
MIT S.B. 1973

5. Charles H. Townes
Physics, 1964
MIT Provost 1961-61

Edit View

Type:	Timeline
Label:	
Start property:	nobel-year (forward) □ Use advanced expression:
End property:	 □ Use advanced expression:
Top band height:	75
Top band unit:	year
Top band pixels per unit:	200
Bottom band height:	25
Bottom band unit:	decade
Bottom band pixels per unit:	200
Timeline height:	400
Show summary:	☑
Show header:	☑
Show footer:	☑

Timeline View Reference

Figure 6: Using the Dido WYSIWYG editor to modify an exhibit. The document is shown in editing mode, with a dialog open for editing a view.

other HTML document or Exhibit. He can also use the built-in editor to modify the text contents of the document, the data being displayed in the Exhibit elements, or the Exhibit elements themselves. When finished, the user can simply save the HTML document in order to persist the changes he has made to the document, the data, and its visualization.

Dido can be used like any other document. The user can store the document anywhere in his file system, or check it into a version control system. He can email it to a friend or post it on a newsgroup. No Internet connectivity is needed to make use of its interactive features.[1] As no data is stored "in the cloud," the user can be confident in the privacy of confidential data.

6 Deployment Experience

Exhibit has been available for public use since 2007. According to access logs, roughly 1800 domains have made use of the exhibit library hosted at simile-widgets.org. Meanwhile, various companies have installed their own local copies of the library (to protect data confidentiality) but we have no way to count them. A healthy community of users participates in the simile-widgets Google group.

[1] The map view is an exception, as it makes use of map services such as Google maps.

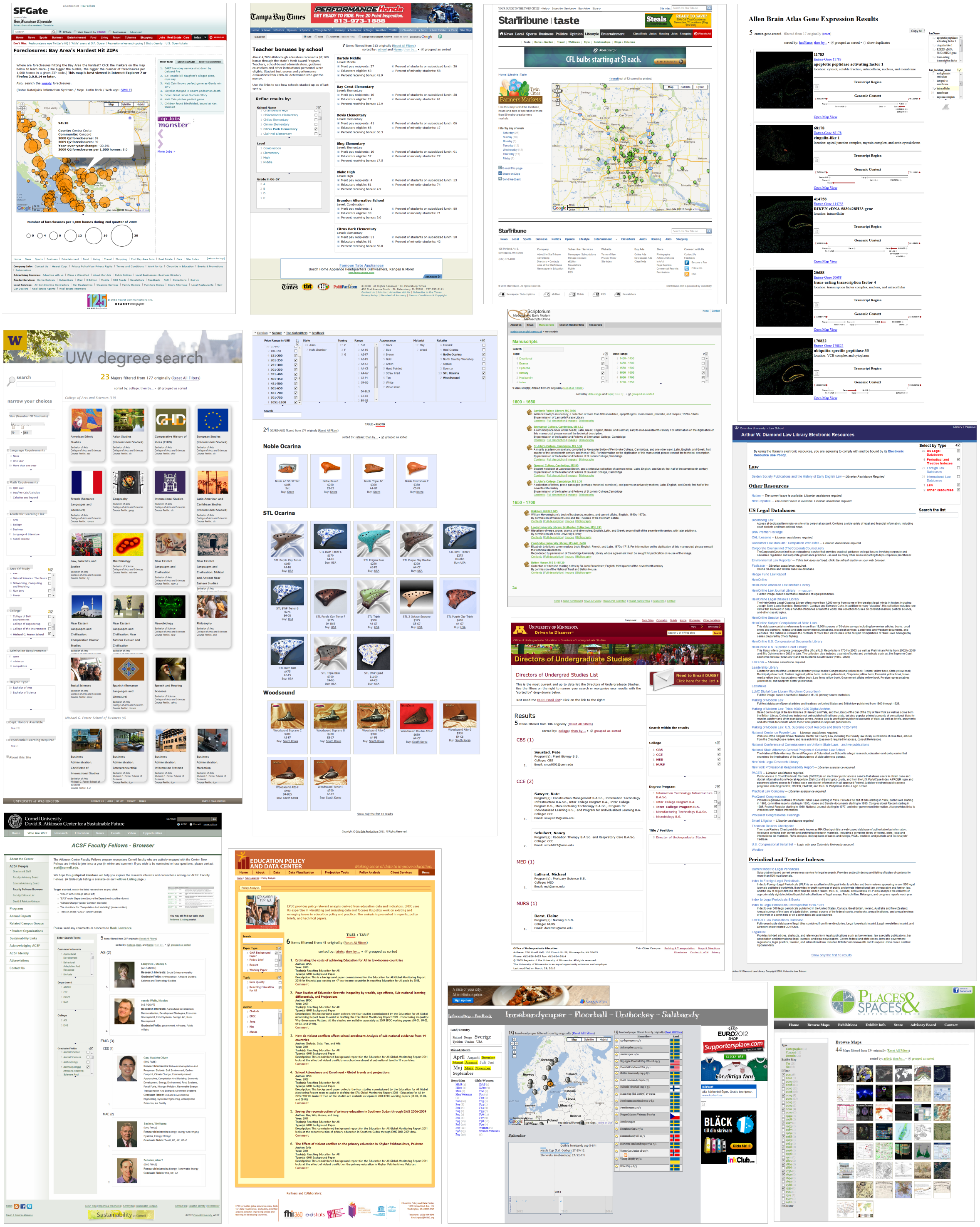


Figure 7: Exhibits in the wild. From top left: Foreclosure rates in the Bay Area (San Francisco Chronicle), Teacher Bonuses (Tampa Bay Times), Farmers Markets (Minneapolis Star Tribune), gene expression data from the Allen Brain Atlas, Degree search at the University of Washington, an Ocarina Catalog, an archive of early manuscripts at Cambridge University, Columbia Law Library electronic resource catalog, University of Minnesota list of academic advisors, faculty directory for Atkinson Center at Cornell University, publication archive for the Education Policy and Data Center, floorball sessions in Northern Europe, a catalog of maps.

6.1 Usage Domains

Even a quick inspection shows that Exhibit presentations are diverse (some appear in Figure 7). Scientists use it to present data sets such as gene expression data, hobbyists to show historical collections such as breweries and distilleries in Ontario from 1914-1915, libraries to present collections of available resources, academics to organize their own or their group's publications or research data, and merchants to present catalogs. It is clear that many of the data types presented have no natural, existing content carrier on the web. Even for those that do, the Exhibits include unusual attributes that would not be available in the standard side. Exhibit is clearly meeting a need which is not being met by other existing tools and sites.

Some newspapers have made use of Exhibit, presenting articles on teacher merit pay and school ratings, pension rates for public employees, farmers' markets in Minnesota and foreclosure rates in the San Francisco Bay Area (see Figure 7). While most offer some basic faceted browsing, the last is notable for presenting only a map. The same map could have been constructed directly using the Google maps API. However, that would have required writing (or leveraging existing) Javascript code to invoke and instantiate it from the data; Exhibit replaces that effort with a single `<map>` tag.

6.2 Keeping it Simple

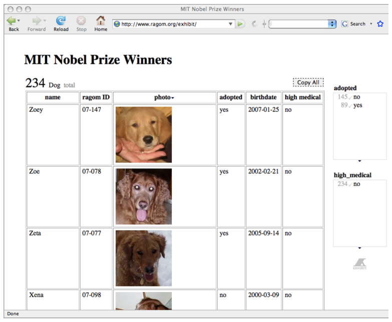
Examination of the published exhibits shows that most users are not leveraging the full power of the data model. Exhibit offers a general entity-relationship data model. Properties of an exhibit item can have other items as properties, and queries can "chain" through these relationships. for example, a person might have a "parent" relationship that refers to another person, and a facet might filter on the birth country of a person's parent. In an exhibit, such a chained expression would be represented using `ex:expression=".parent.birthCountry"` in the facet tag. However, the vast majority of exhibits on the web do not leverage this power. Their data model consists only of a single table, with all properties binding to simple literal (number, date, or text) values.

There are two plausible explanations for this limited data richness. One is that more complex models are rarely needed. Another is that they are needed, but that they are too complicated for users to think about or implement. Our experience with Exhibit doesn't help to choose between these two options. One might jump to the conclusion that if nobody is using the richer model then nobody wants it. But discovering that this richer model is available requires either reading the tool's documentation (which is rarely done) or discovering and being inspired by existing examples of the richer usage (which don't exist, a typical chicken-and-egg problem). And even if users discover it, some challenge in using it may discourage users from making the effort or complaining to us about their failures.

The first interpretation does raise the possibility that if it made usage easier, Exhibit could comfortably sacrifice its more powerful data model for a flat tabular one. It isn't clear that this change would actually allow us to simplify Exhibit's description language, which is already quite clean. However, switching to a table-based database implementation could conceivably offer significant performance improvements, allowing exhibit to scale to larger data sets.

6.3 Learning by Copying

Exhibit's design means that Exhibits are portable: someone who sees an exhibit page an likes it can generally download that page (and its accompanying data page) to get her own copy of the exhibit, which she can then modify to serve her own needs. This copy-and-modify paradigm was a common way for new authors to begin creating HTML content in the early days of the web, and remains important today [BGL$^+$09]. Rather than deeply understanding HTML and creating pages from scratch, authors could download an appealing page and modify only key portions, learning the HTML as they



MIT Nobel Prize Winners

234 Dog total

name	ragom ID	photo	adopted	birthdate	high medical
Zoey	07-147		yes	2007-01-25	no
Zoe	07-078		yes	2002-02-21	no
Zeta	07-077		yes	2005-09-14	no
Xena	07-098		no	2000-03-09	no

adopted

145	no
89	yes

high_medical

234	no

Figure 8: From Nobel prizes to dogs

went. We do have some evidence of this copying behavior in Exhibit. An early version of the Columbia Journalism Fellows page was a near-exact copy of one of our example exhibits, the MIT CSAIL Principle Investigators page. Another exhibit of a collection of dogs showed, through a failure to update its title, that it had been copied from our example exhibit of MIT Nobel prize winners (see Figure 8).

However, a barrier to this spreading-by-copying has been Exhibit's invisibility. Someone visiting an Exhibit has an experience very similar to those provided by database-backed, template driven web sites. There is no visible sign that the entire functionality is contained in the page itself, and that the visitor could duplicate that functionality on their own data by downloading and modifying the page. Thus, users generally need to learn about Exhibit through some other channel before they begin the experiment of creating their own.

7 Discussion

7.1 The Semantic Web? Linked Open Data?

From the beginning, Exhibit was envisioned as a stepping stone to the Semantic Web. Its design causes people separate their data from the visualization of that data, and to place the data itself at an easily accessible URL. At the same time, Exhibit refuses to let the best be the enemy of the good. To keep authoring simple, Exhibit steered towards simpler data representations, preferring JSON and CSV over the more powerful RDF. This prevents authors from making use of RDF's schema annotations or inference rules. There is no easy way for an Exhibit author to indicate that their "country" property is the same as a "location" property appearing in some other Exhibit.

A similar move towards simplicity can be observed in the *Linked Open Data* community, which has dropped discussions of inference and focused on encouraging authors to use the same URLs to name entities in different data repositories. But Exhibit steps back even from this, settling for *Open Data*. Exhibit authors are *permitted* to use URLs to identify their items, allowing linkage of entities across different Exhibits, but are not *required* to do so. Making URLs a requirement would add to the effort of publishing data, possibly deterring it. While URLs are valuable to (data) consumers, they offer no clear benefit to the author.

Our reasoning for this approach is that opening data is a necessary precursor to anything more sophisticated, and that it often provides the bulk of the benefit. In the absence of open data, consumers must either create scrapers that extract information from unstructured sources (which requires programming skills and generally yields imperfect results) or manually extract/transform the information about every item in a data set. Substantial effort is required.

In contrast, given a structured data set, the biggest problem a user is likely to encounter is understanding the intent of the various properties. While RDF offers a sophisticated language for describing these semantics, having the user do so by eyeballing the tens of properties that make up a typical data set is no great burden.

Even if a user's goal is to "mash up" multiple sources of data, simply having it structured leaves him much better positioned. Once the user understands the semantics of properties, he can manually rename them, or split/merge columns, to align the schemas of distinct data sets. The other challenge, entity resolution often boils down to string matching on names or other primary-key properties.

In summary, I believe that we should aim low, sacrificing any non-essential requirements in favor of making it as simple as possible to publish structured data. Should this work, and a broad structured data ecology emerge, it will naturally generate demand for improvements, such as better alignment of entities and properties. However, until we reach that point, imposing such requirements can only deter progress.

7.2 Exhibit and Microformats

There's been significant recent work on *microformats*, a standard for representing data unambiguously in HTML. The HTML 5 standard has introduced elements (particularly the "data-" prefix for attributes) to support the usage of microformats in valid HTML 5. Various browser extensions have been created that are able to recognize and extract this data from browsers in useful ways. For example, an extension recognizing the vCard microformat can easily incorporate the personal information presented in a web into a contact management application, while a recognizer for the hCalendar microformat can pull event information into a calendar program. It would be natural and easy for Exhibit to take microformats (and microdata, and RDFa) as yet another data representation that it should parse.

More ambitiously, Exhibit itself can be seen as a prototype for a *visualization microformat*, a standard for *describing* visualizations separate from *implementing* them. This can be seen most directly in Exhibit's approach to the map view. Exhibit offers three map extensions, one based on Google Maps, one based on Microsoft Maps, and one based on Open Streetmaps. The author determines which one is used by choosing which library to link. The exhibit description of the map, using a `<map>` tag, is the same in all three cases. This approach can be generalized. There are already multiple implementations of timelines, of various charting libraries, and of thumbnail galleries. If these were all able to parse the same visualization microformat, then they could be used interchangeably to produce visualizations. The same could be done for visualization implementations that have not been invented yet, creating opportunities for collaborative competition.

The usual challenges to defining standards pertain. The three Javascript mapping APIs that Exhibit currently "wraps" all offer different interesting options. A rigorously standard microformat would only offer the options that are available in all mapping extensions. But this would be in tension with the desire of each map "vendor" to expose the special capabilities of their mapping extension. We may see the emergence of a situation like that in today's browsers' competition over CSS, where each introduces its own non-compliant vocabulary elements which compete to enter the standard.

Taking this to an extreme, one could imagine incorporating visualization as just another aspect of CSS—to add parameters like `display: map` to the CSS vocabulary, and expect browsers to contain to the logic necessary to create such visualizations. However, the sheer variety of visualizations argues against this approach; it seems impossible to define a comprehensive characterization of all visualizations that would need to be implemented by the browsers.

Nonetheless, as with microdata, it seems plausible to define an "extensible standard." Microdata describes a syntax that can describe any type of data, then leaves it to a subcommunity to decide on the specific properties of a given data type that should be expressed using the syntax. Similarly, one could imagine numerous specialized map visualizations emerging that extend the basic map-visualization type; rendering tools that didn't understand the particulars of one of these specialized visualizations could still fall back on a more basic map that accomplished a good part of the visualization goal.

One might ask why competing visualization providers would ever wish to agree on a common standard for visualizations. While the answer is unclear, what is clear is that it does sometimes happen. The standardization of HTML has led to the emergence of competing rendering engines such as WebKit and Gecko that are continuously pushing each other to improve. At schema.org, many of the largest information companies are collaborating in the specification of formats and schemas for describing specific types of data. Presumably, each company perceives more benefit to competing on a large shared playing field than they do in dominating part of a fragmented domain.

7.3 The Durability of Standards

In addition to providing opportunities for beneficial competition, separating visualization description from implementation offers the possibility of enhanced longevity for our content. A static HTML document written in the 1990s still renders reasonably well on modern browsers. Meanwhile, maps implemented in Javascript using early versions of Google's or Microsoft's mapping APIs, or relying on startups that have since failed, no longer function. We continue to make use of standards defined decades ago—SMTP, TCP, VNC, AES, RSA—while our software systems become obsolete within a few years. As technology changes, it is much easier to write new software for old standards than it is to maintain or replicate large software systems.

Today's web is filled with cloud-based services providing an amazing variety of rich, interactive visualizations. But how many of those services will still exist in five years? Many cloud services offer mechanisms for exporting a user's data in familiar standard formats. Meanwhile, the only way to export a visualization is to take a screenshot. A standard for visualizations could offer a mechanism for exporting visualizations that would let them outlast the sites where they were created. Given the ongoing attempts by competitors to innovate, we might expect the visualization export to lose some of the particular attributes it had on a given cloud service, but we could still hope for far higher fidelity than we currently get from a screenshot. In particular, we could hope to export some of the interactivity that we have come to expect of modern visualizations.

7.4 Performance Optimization

If our approach is truly able to capture a great deal of the data interaction authors and readers want to support and experience on the web, then creating a restricted vocabulary opens the opportunity for significant performance optimizations. Behaviors that are represented in HTML (and CSS) are executed by browsers using native code that is far faster than Javascript libraries. If in-page data, facets, views and lenses become a standard part of the HTML vocabulary, there are similar opportunities for performance optimization. Page data could be stored in a native SQL-style database, which would dramatically improve the performance of the filtering specified by facets. An appropriate compressed representation of a page's data could also be transmitted and loaded far faster than our current representations. Lens templates could be rendered quickly using an engine similar to the XSLT engine already built into browsers. I believe that Exhibits of tens of thousands of items could easily be manipulated in user-interactive time given such native implementations.

8 Conclusion

This article has argued that a great deal of interactive data visualization on the web can be captured in a relatively simple vocabulary of data, views, facets, and lenses, and that this vocabulary can in turn be effectively represented by a small number of new HTML tags. Such a problem simplification offers many advantages we've discussed above, including better separation of content and presentation, application independence, longevity greater than that of cloud services, and reduced load on servers.

Perhaps most importantly, doing so transforms what has until-now been a visualization *programming* problem into a visualization *authoring* problem. This in turn makes it possible to create authoring tools that enable end users to create visualizations rivaling many of those programmed on professional web sites. With a fully fleshed out set of these end-user data-oriented tools, it may be possible to spark a simplification and democratization of structured data authoring on the web that could imitate the web's original, text-based revolution.

References

[BGL$^+$09] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1589–1598, New York, NY, USA, 2009. ACM.

[BMHK10] Edward Benson, Adam Marcus, Fabian Howahl, and David R. Karger. Talking about data: Sharing richly structured information through blogs and wikis. In *International Semantic Web Conference*, pages 48–63, November 2010.

[Con04] The World Wide Web Consortium. Resource Description Framework. `http://www.w3.org/RDF/`, November 2004.

[DVWK08] Catalina M. Danis, Fernanda B. Viegas, Martin Wattenberg, and Jesse Kriss. Your place or mine?: visualization as a community component. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 275–284. ACM, 2008.

[HKQ02] David Huynh, David Karger, and Dennis Quan. Haystack: A platform for creating, organizing, and visualizing information using RDF. In *Semantic Web Workshop at WWW2002*, Honolulu, Hawaii, May 2002.

[HMK07] David Huynh, Robert Miller, and David R. Karger. Exhibit: Lightweight structured data publishing. In *WWW 2007*, pages 737–746, May 2007.

[KOL09] David R. Karger, Scott Ostler, and Ryan Lee. The web page as a wysiwyg end-user customizable database-backed information management application. In *UIST '09: Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 257–260. ACM, October 2009.

[KPP$^+$12] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Profiler: integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 547–554, New York, NY, USA, May 2012. ACM.

[MLF95] Thomas W. Malone, Kum-Yew Lai, and Christopher Fry. Experiments with oval: A radically tailorable tool for cooperative work. *ACM Transactions on Information Systems*, 13(2):177–205, April 1995.

[NCW^{+}07] Stefan Nusser, Julian Cerruti, Eric Wilcox, Steve Cousins, Jerald Schoudt, and Sergio Sancho. Enabling efficient orienteering behavior in webmail clients. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, pages 139–148. ACM, 2007.

[PBKL06] Emmanuel Pietriga, Chris Bizer, David Karger, and Ryan Lee. Fresnel: A browser-independent presentation vocabulary for rdf. In 5^{th} *International Semantic Web Conference (ISWC)*, pages 158–171, November 2006.

[Pie06] Emmanuel Pietriga. Semantic web data visualization with graph style sheets. In *Proceedings of the 2006 ACM symposium on Software visualization*, SoftVis '06, pages 177–178, New York, NY, USA, 2006. ACM.

[VHAA11] Amy Voida, Ellie Harmon, and Ban Al-Ani. Homebrew databases: complexities of everyday information management in nonprofit organizations. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 915–924, New York, NY, USA, 2011. ACM.

[YSLH03] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, pages 401–408, New York, NY, USA, 2003. ACM.