Standards Opportunities around Data-Bearing Web Pages

David Karger

INTRODUCTION

The web dramatically simplified, and thus democratized, the authoring and sharing of information. 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.[1]

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. They 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 creating to pursue this objective. 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.

[1]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.

Proceedings of HCIR 2012, Cambridge MA, Oct 4-5, 2012

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 also 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. Updates will immediately show up in faceted browsing interfaces, without the author needing to rebuild "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. They 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 for 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 [4]. 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 [3]. If they publish it to the web at all (often they do not) they are limited to plain tables [1], instead of the rich interactive visualizations offered by professional sites.

Approach

To address these problems, I propose to 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 structure 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 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 visual 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 map 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.

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 content differently from a desktop browser. A browser for the blind can use the structural description to prepare an *audio* presentation of the content and leverage the structure to help its user navigate the content.

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. He is in complete control of how the visualization elements interleave with the text and structural content of the page. He can place views side by side or at widely separate parts of the page. He can place facets in whatever relation they like to the data views. Authors thus have full control of the presentation of their information narratives.

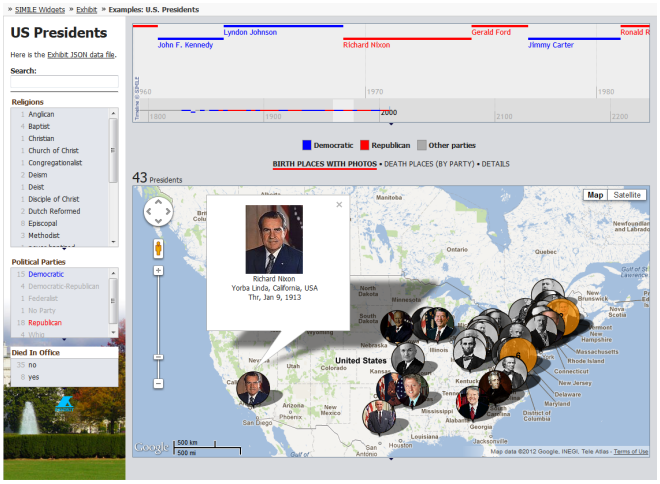

Figure 1. An exhibit of U.S. Presidents. A timeline view (color-coded by presidential party) is shown above a map of birthplaces. The left pane contains facets for filtering by religion, party, and whether they president died in office, as well as text search.

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 we sketch the Exhibit language and library; more details can be found in our paper on Exhibit [2]. A sample Exhibit can be seen in Figure 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 conveyed to Exhibit in a variety of formats: JSON, a comma- or tab-separated values file with one row per item, a Google spreadsheet, or various other formats.

To indicate their data, the author creates a reference to their data file using a standard html `link` tag, 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 directly in the presentation document.

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. The individual items to be 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 `div` tag. The author then specifies the particular type of view they want 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.

Many of the views take additional attributes specifying their configuration. For example, the map view's `ex:latlng` attribute specifies 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.

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 to name 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.

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.

Even a quick inspection shows that Exhibit presentations are diverse (some appear in Figure 2). 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.

EXHIBIT AND MICROFORMATS

There's been significant recent work on *microformats*, a standard for representing data unambiguously in html. The html5 standard has introduced elements (particularly the "data-" prefix for attributes) to support the usage of microformats in valid html5. 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 "hcal" can pull event information into a calendar program. It would be natural and easy for Exhibit to take microformats (and microdata, and rdf-a) 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 interchangeable to produce visualizations.

The same could be done by visualization implementations that have not been invented yet, creating opportunities for competition and providing valuable longevity. The standard of html has led to the emergence of competing rendering engines such as WebKit and Gecko that are continuously pushing each other to improve. 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.

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.

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.

Figure 2. 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.

REFERENCES

1. Benson, E., Marcus, A., Howahl, F., and Karger, D. R. Talking about data: Sharing richly structured information through blogs and wikis. In *International Semantic Web Conference* (Nov. 2010), 48–63.

2. Huynh, D., Miller, R., and Karger, D. R. Exhibit: Lightweight structured data publishing. In *WWW 2007* (May 2007), 737–746.

3. Voida, A., Harmon, E., and Al-Ani, B. 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, ACM (New York, NY, USA, 2011), 915–924.

4. Yee, P., Swearingen, K., Li, K., and Hearst, M. Faceted metadata for image search and browsing. In *Proc. ACM CHI Conference on Human Factors in Computing* (2003).