The Web Page as a WYSIWYG End-User Customizable Database-Backed Information Management Application

David R. Karger*
karger@mit.edu

Scott Ostler
sbostler@gmail.com

Ryan Lee
ryan@voccs.com

MIT CSAIL
32 Vassar St.
Cambridge, MA 02139

ABSTRACT

Dido is an application (and application development environment) in a web page. It is a single web page containing rich structured data, an AJAXy interactive visualizer/editor for that data, and a "metaeditor" for WYSIWYG editing of the visualizer/editor. Historically, users have been limited to the data schemas, visualizations, and interactions offered by a small number of heavyweight applications. In contrast, Dido encourages and enables the end user to edit (not code) in his or her web browser a distinct ephemeral interaction "wrapper" for each data collection that is specifically suited to its intended use. Dido's *active document* metaphor has been explored before but we show how, given today's web infrastructure, it can be deployed in a small self-contained HTML document without touching a web client or server.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Documentation, Human Factors

Introduction

This note describes Dido, a *Data-Interactive DOcument* that lets end users *author* (not program) web-page "applications" specifically suited to their own information management needs. Instead of forcing their data into a partially-suitable application (the first author uses a music manager to run a folk dancing session, shoving choreographer into the artist field, dance type into genre, and difficulty into the comments), or settling for a spreadsheet's generic tabular interface, users can wrap any data collection in a data- and task-specific interface.

Dido is an *active document* [7] with a look and feel similar to those delivered by typical content-management web sites such as Flickr (photos), Amazon (products), CNET (reviews), LinkedIn (people), and Epicurious (recipes). An active document incorporates elements that sort, filter, or otherwise manipulate the information being presented, updating the document on the fly to reflect the manipulation. Data edits in the document can change data stored on the server.

The active document is a pervasive and well-understood metaphor on today's web. But these documents' dynamic UIs, server-side databases, and AJAX connections are generally *programmed* by professionals. In contrast, Dido documents, including all their active elements, are authored by end users in a WYSIWYG document-editing interface similar to those used for authoring static HTML documents.

Dido's active document is not just an interface metaphor. The data being managed, the UI for interacting with it, and the "metaeditor" for authoring the interaction UI are all stored in one HTML document. No deployment or installation is needed—the document "just works" in any modern web browser. Authors can share their active documents freely without worrying who has the right application installed. The in-document data store eliminates the server and encourages users to treat the entire "application" as a document that can be copied, edited, emailed, placed in a version control system, or published to the web. With Dido, an application is not a heavyweight object that owns data; rather, each data set comes wrapped in a light and flexible application "skin."

Obviously there are limits to the complexity of applications typical users can author. We focus on so-called *CRUD* applications that let users (C)reate, (R)ead/Visualize, (U)pdate, and (D)elete items. Contact managers, todo lists, recipe managers, and photo albums fit this class. CRUD is the core of widely deployed *Content Management Systems* such as Drupal, Nuke, Sharepoint, Django, and Semantic Mediawiki.

This note combines two theses. First, that there is a common set of CRUD interactions that can and should be stripped of their programmer-oriented complexity and made part of the common vocabulary of document authoring, allowing end-users to customize or create their own task-specific "applications." Second, that the ubiquitous web browser contains all the machinery needed to support these commonplace data management tasks, making them trivial to implement and deploy in standard web documents. We elaborate on these ideas and discuss their relation to previous work after a walk through the Dido system.

Easy authoring and universal compatibility resolve the two

*This work was supported by a grant from the National Science Foundation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
UIST'09, October 4-7, 2008, Victoria, BC, Canada.
Copyright 2009 ACM 978-1-60558-745-5/09/10...$10.00.

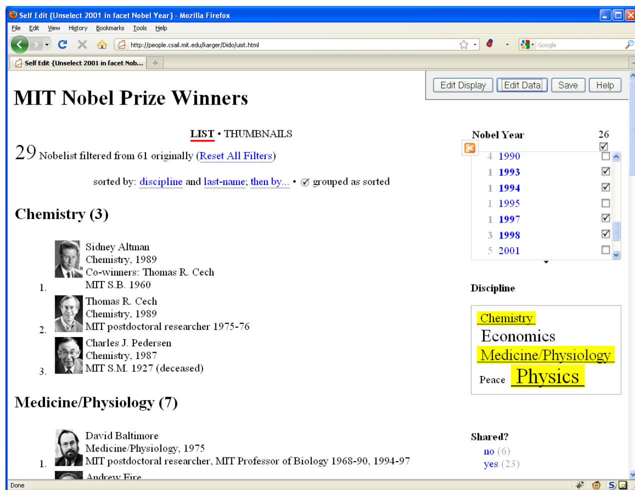

Figure 1: Visualizing data in Dido. Three disciplines and four visible years have been selected as filters.

biggest concerns with active documents. So consider broad adoption. Instead of publishing a user study as a (noninteractive, opaque) PDF document referring to a (raw, mysterious) data file, we could publish one active document containing both text and data in a meaningful interactive visualization. Had this paper been published as HTML, its static figures could have been interactive demonstrations of Dido.

Walk-through

A user opens a Dido document in his or her browser, downloading it from a web site or opening a local copy. Figure 1 shows a typical example: a list of MIT Nobel prize winners, grouped by discipline and sorted by last name. The user can change the sort by clicking in the "sorted-by" description. A line above allows the user to switch from the detailed *List* currently selected to a grid of photo-only *Thumbnails*. On the right are three different styles of *facets* used to filter the list by year, discipline, and whether the prize was shared.

This kind of interactive document is typical of today's content-bearing web sites. It presents a collection of *items* (Nobelists) with *properties* (name, Nobel-year, or discipline) that have *values* for each item. Examples include photos on Flickr (creator and date), books on Amazon (genre and price), and recipes on Epicurious.com (cuisine and ingredients). These sites use their items' properties to fill templates showing individual items, to compute aggregate views of item collections, and to support sorting and filtering via faceted navigation [9].

The Dido visualization uses our *Exhibit* framework [5] which abstracts interactions commonly found on today's web sites. A *lens* is a template for rendering an item—an HTML fragment with slots to be filled by the values of specified properties. A *facet* is a widget that filters items based on the value of some property. A *view* shows the entire (filtered) collection in some way: for example a map view plots items on a map using the coordinates specified in some property, while a list view shows the items in (sortable) order using lenses. Exhibit offers several views—lists, thumbnails, maps, timelines, scatter plots, and pivot tables—and several filtering facets—lists, sliders, tag clouds, and numeric ranges.

Exhibits are described by a data file holding the items and an HTML file with special tags added where Exhibit widgets should appear. Exhibit's Javascript fetches the data file, then interprets the Exhibit tags to provide data interaction when the page is viewed. The explicit properties-and-values data model means each widget can be configured without programming, by specifying which properties play which roles in the widget, much as one specifies which columns of a spreadsheet should be used as data for a particular chart. For example, the Map view specifies which properties contain the latitude and longitude needed to plot each item, while the list view specifies the initial sorting property.

Exhibit has been adopted by several hundred users, including a few newspapers, to publish interactive visualizations such as a Vegan guide to Glasgow restaurants, caves in Italy, a history of classical music composers, a list of cases at the European Court of Human Rights, conference proceedings and programs, and many others (a list can be found at `http://simile.mit.edu/wiki/Exhibit/Examples`). This suggests that Exhibit's interaction vocabulary suffices for some data presentation tasks. Exhibit suffers from two limitations, however. First, it offers read-only visualization of data stored on the author's web site; the reader is a passive consumer. Second, the author is expected to be comfortable editing both the raw data file and raw Exhibit XML tags.

Dido packages Exhibit's data and presentation vocabulary into a single file and delivers it to the reader with an editing framework that helps the reader become the (WYSIWYG) author as well. If the user clicks the "Edit Data" button on the top right, the editable values of any items being shown are highlighted. The user can click on any such value; it will convert to a text box for editing. Clicking outside the textbox completes the edit and returns the textbox to its original format. There are also buttons for creating and deleting whole items. The user can continue to filter and otherwise interact with the data while it is in editable form.

This form of direct editing is simple to implement because of Exhibit's lens-based architecture. Since items are rendered via HTML lens templates that specify how content should fill in the blanks, we invert those templates to infer the edit target "blank" from a user's click.

A user can change the *visualization* by clicking "edit display" on the top right. This shifts Dido to meta-editing mode. We invoke a WYSIWYG editor on the entire Exhibit. Individual data items are no longer editable; rather, the page is. The author can click anywhere in the document to add static content, or use the editor toolbar to change the layout (adding tables, marking paragraphs, etc.) and the formatting (choosing fonts and sizes, etc.).

Beyond the static layout, the user can modify the interaction. In meta-editing mode the user double clicks views and facets to open configuration dialog boxes that specify the wiring between widgets and data. For example, in the list-view dialog, users can specify the property used for the default sort of the items in the list. In the timeline view, they can specify the properties whose values define the start and end times marking each item's interval on the timeline. Richer expressions

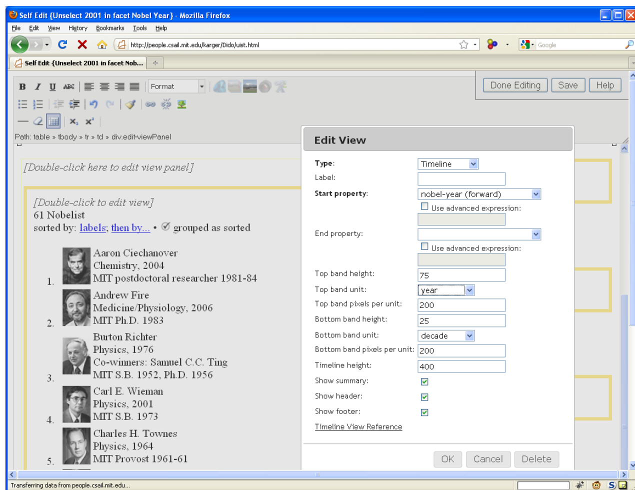

Figure 2: Editing the Visualization to add a timeline

are possible as described in our previous paper on Exhibit [5]. Previews of the views and facets are shown in place.

The lens templates, which were invisible in data manipulation view, become visible and editable. The user can double click one to configure its global properties, such as which type of item that lens can be used to display. The user can change the layout of the lens through standard WYSIWYG editing, or can double click on the content elements (fill-in-the-blanks) to configure which properties of an item should be rendered at various places in the lens. By adding new properties to the lens, the user implicitly extends the schema.

The editor toolbar has been extended with buttons to create new Exhibit elements: facets, views, and lenses in the main body, and content elements in the lenses.

Once the user is satisfied with their changes, they can return to the standard view by clicking "Done Editing" at the top. If they click the adjacent "Save" button, the browser will open a dialog to get permission from the user and will then save the document to a file with its modified data and visualization.

Implementation

Dido incorporates Javascript from several systems. We modified Exhibit (http://www.simile-widgets.org) to support data editing and to load and store its data (as a JSON block) and presentation (as an HTML block) in the same document. We incorporated TinyMCE (http://tinymce.moxiecode.com), a WYSIWYG HTML editor, and augmented it with buttons and dialogs for editing Exhibit widgets. Dido's file-save code comes from TiddlyWiki (http://www.tiddlywiki.org/), whose developers have ensured that it works on most browsers even though there is no standard file API. The dialogue interactions used to configure widgets, and many additional processing steps, are simplified by our use of the jQuery DOM manipulation and UI libraries (http://jquery.com/).

Because so much of Dido's functionality is built into modern web browsers, Dido is small. We wrote 4000 lines of HTML and Javascript and bundled them with 2Mb of Javascript from the robust open source libraries just described.

Discussion

Dido is an *Active Document* [7, 1], a document that incorporates interactive behaviors typical of applications. Terry and Baker [7] assert that "such applications may be easier to build than their traditional counterparts because they can take advantage of the capabilities of a document editor. In addition, they may be easier for users to learn because of the user's familiarity." Active-document ideas appeared in products such as Interleaf Active Documents [3] and Apple's Hypercard [2]. But an adopter of these tools had to learn a new metaphor and work in a new system. And since most computers did not have these tools, he could not count on others' being able to read the active documents he produced.

The passage of time has changed this situation. Today every HTML/Javascript web page is an active document. Users navigating today's web *expect* the documents they view to offer sorting, filtering, multiway visualization and even editing of their contents. And universally deployed web browsers offer all the machinery necessary to support these interactions. The boundary between document and application has blurred. Thus, it is time to reconsider active documents.

A key question is how active documents can be *authored*. Early systems [7, 2, 3] all exposed *scripting languages* in which authors were expected to program behaviors for the active document elements. Javascript is the latest such language. Dido shows that a broad class of behaviors can be defined without scripting by surfacing an explicit property-values data model and binding UI widgets to its properties.

The web currently exhibits substantial uniformity in interfaces for content interaction. While it may be disrupted by the introduction of richer interface technologies such as Flash or Silverlight, there is a present opportunity to capture this uniformity in a visual vocabulary that can be authored by end users. The HTML 5 standards body has proposed an API for an in-page property-value data model [4]; Dido suggests we could define HTML-standard tags describing interactions with that data (sorting, faceting, etc.), just as we currently have tags for tables and pictures.

Dido reconsiders what "applications" are. Historically, they have been large complex objects developed by professionals. Each followed a specific schema. Installing one was a significant operation. Many were available only to certain users. In contrast, a Dido application is light. There is no installation—it arrives attached to its data set, ready to use in any browser. It can be seen as nothing more than a simple skinning of the data, like a cascading style sheet. Because they are so easy to create, there could be as many different visualizations as there are data sets, with each tuned to a single use. Dido applications are simply HTML documents that can be tossed into a user's file space. If a user sees a content presentation he likes, he can download it and insert his own data. There is no need to decide whether to upgrade a Dido application—a user can keep both versions. He can make copies freely, put them in version control systems, and generally treat them with the sloppiness typical of documents, rather than the care typical of applications.

Dido is scoped to CRUD applications. Some applications

offer much more complex operations over their data—an accounting program reconciles checks, while a project-planner computes critical paths. But even they are *mostly* CRUD. Dido might "link to" certain richer computations: since an `mms:` url launches a media player and a `mailto:` url an email program, Dido could perhaps be used to make a media manager with email-sending capability.

For the researcher, the tiny amount of code forming Dido shows just how simple it is to create an end-user-authorable active document framework in today's web environment. There is now an opportunity, at very little cost, to explore a variety of active document frameworks different from Dido, and discover which ones are most effective for end users. Unlike typical web applications', all of Dido's code can be found in its documents, which streamlines its creative reuse in research and open-source settings.

To explore its limits, we pushed hard on the idea of "all in one document." But there are many middle grounds. Dido's metaeditor, visualization and data are distinct elements that could be unbundled in various combinations. The metaeditor could load or save separate visualization/data bundles. A visualization could read/write its data from/to a different file or some cloud store such as a Google spreadsheet. Whole Dido documents could also be managed in the cloud—the active document is still an effective interface-editing metaphor.

Despite the popularity of the cloud, placing all content in one local document does offer benefits. A server adds complexity and a point of failure to the system, and demands connectivity. While cloud applications are accessible everywhere, there is no guarantee that their data can easily be extracted and combined with data from other web applications. Dido's data moves in its pages to wherever it is needed, and is easy to extract and combine [5, 6]. Another important benefit of all-in-one-file is that Dido can be used by an individual to manage sensitive data that could not safely be stored in the cloud. Dido data can be shared on a private web server or emailed on a secured channel. In contrast, many organizations have data they are unwilling to risk placing on a publicly accessible cloud application where it might leak. On the other hand, some functionality *must* reach outside the document: Dido's map view invokes Google Maps' web service, as a world map is too large to fit in the document.

Dido is only a proof of concept and can be enhanced. For example, all data editing at present happens through text fields. There are better widgets for editing specific data, such as a checkbox to set a boolean value or a dropdown menu to select from among a fixed set of possibilities. Such widgets are commonplace in web forms, and could be added to Dido by adapting TinyMCE's WYSIWYG form editor. There are opportunities to blur the data-editing and presentation-editing modes: a facet could be inserted while editing data, and changing the format of a displayed item could map to a format change on the underlying lens. Dido (through Exhibit) already supports copying its contained data to the clipboard for use in other applications; we plan to add a corresponding data-paste, and to extend it to allow copy/paste of Exhibit widgets or whole visualizations. Users could then create *mashups* by pasting two Dido documents into one.

Several present-day tools are related to Dido. Space precludes more than a brief mention. ManyEyes [8] is a data visualization web site. It focuses on letting users upload (not manage) data sets and create visualizations on the web. TiddlyWiki is an entire wiki implemented via Javascript inside one web page; the wiki is persisted by saving the file. Its contents are text, not structured data. Filemaker (http://www.filemaker.com/) is a data management desktop application for end-users; Dabble DB (http://dabbledb.com/) is a similar system on the web. Like Dido, these systems let end users create data sets and configure interactions with them. However, they offer a traditional application rather than an active document metaphor. Like early active document systems', Filemaker's limited deployment makes it harder for people to share information. ManyEyes' and Dabble DB's web presence means anyone can access them; this has pros and cons we have already discussed.

Conclusion

Instead of warping their data to fit rigid applications, users should warp applications to fit their data and tasks. The evolution of the web has made the *active document* a timely candidate for supporting this goal. Dido shows how to address two major challenges for active documents, authoring and deployability. A properties/values data model supports WYSIWYG authoring of CRUD interactions, and placing the entire framework in a web page makes deployment trivial. While it draws on old ideas, Dido shows how today's Web infrastructure can be used to make end-user customization via active documents a commonplace approach to data and application management.

`http://projects.csail.mit.edu/exhibit/Dido/` offers a download of Dido.

REFERENCES

1. Eric A. Bier. Embeddedbuttons: supporting buttons in documents. *ACM Trans. Inf. Syst.*, 10(4):381–407, 1992.
2. Apple Computer. Hypercard software & user's manual, 1987–1998.
3. Paul M. English and Raman Tenneti. Interleaf active documents. *Electronic Publishing*, 7(2):75–87, June 1994.
4. Ian Hickson. Web storage. Technical report, World Wide Web Consortium, April 2009. `http://www.w3.org/TR/webstorage/`.
5. David Huynh, Robert Miller, and David R. Karger. Exhibit: Lightweight structured data publishing. In *WWW 2007*, pages 737–746, May 2007.
6. David Huynh, Robert Miller, and David R. Karger. Potluck: Data mash-up tool for casual users. In 6th *International Semantic Web Conference (ISWC)*, November 2007. to appear.
7. Douglas B. Terry and Donald G. Baker. Active tioga documents: an exploration of two paradigms. *Electronic Publishing—Origination, Dissemination, and Design*, 3(2):105–122, 1990.
8. Fernanda B. Viegas, Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matthew M. McKeon. Manyeyes: a site for visualization at internet scale. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1121–1128, 2007.
9. Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *Proc. ACM CHI Conference on Human Factors in Computing*, 2003.