Customizable Application for Personal Information Management

by

Jamie Liu

S.B., Massachusetts Institute of Technology (2005)

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

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author .... [illegible] ....
Department of Electrical Engineering and Computer Science
May 17, 2010

Certified by.... [illegible] ....
~~David R.~~ Karger
Professor, Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .... [illegible] ....
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Customizable Application for Personal Information Management

by

Jamie Liu

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2010, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

As digital content becomes increasingly pervasive in our daily lives, next-generation organizational tools must be developed to help end users curate that information. The information management tools available today are limited in several ways. They are either restrictive in how users define and organize their data, or they offer few options in how users can view and interact with that data. In this thesis, we introduce a new application geared toward the layman end user to help resolve this lack of personalization when managing data. With this tool, users can store any collection of information (ie. to do lists, address book, DVD collection), customize how they want to view and browse that data, and create any number of visualizations for the same data set or overlapping sets of data. For example, a user might want to manage an address book for a collection of all contacts, and manage a different set of items involving all research-related items, including colleagues, in another visualization. This tool allows her to do this without duplicating the overlapping data set for contact information of colleagues, enabling her to make changes to an item in one visualization and see that change applied to any other visualization that includes the same item. Moreover, users can import data from other sources, as well as share their data with others either with a visualization, such that the receiver can interact with the visualization the user has already created, or as raw data, such that the receiver can create his or her own visualization of the data.

Thesis Supervisor: David R. Karger
Title: Professor, Department of Electrical Engineering and Computer Science

Contents

List of Figures

Chapter 1

Introduction

The scope and quantity of digital information has increased dramatically over the past several decades. The internet gave users access to more data, first as static content contained in linked documents, followed by larger quantities of dynamic content displayed in scalable, interactive, database-backed websites, and now in more cases as raw data[16], where information is provided sans presentation. Further, the emergence of "Web 2.0" technologies has given rise to improved social media sharing and organizational tools, and the ability to sort, search, and filter data sets became the status quo.

With the growing amount of accessible data, and the expanding feature set with which software applications enable users find content quickly, people are becoming increasingly reliant on software to manage more of their information. However, existing tools for managing one's own personal data remain limited. Current applications provide inflexible structures for storing and saving information, and/or provide limited means of visualizing and browsing that information, such that users cannot curate their content in a manner most appropriate for their personal needs. We try to resolve this by creating a customizable tool to allow layman end users to visualize the data they care about in ways that are most effective for them. With this tool, users can store any collection of information (ie. to do lists, address book, DVD collection), specify the properties of that data, choose how they want to visualize their collection, and decide what type of filter widgets to add to best browse and retrieve their data.

Users can also safely create any number of visualizations for the same data set or overlapping sets of data.

1.1 Related Work

Today, people must use different software applications to manage their different types of information[1]. We consider two classes of information management tools: applications designed for specific data types ("Data-specific applications") and those designed for a generic set of data ("Generic data applications"). We describe the benefits and limitations of these two categories of applications in this section. However, before we dive into existing applications, let us first first define a persona to help us illustrate possible use cases.

1.1.1 Persona

Graduate Student Sally is a multifaceted person with information to manage for both her professional and personal life, including contacts, tasks, papers and books read. For example, Sally needs to maintain a contact list for everyone she knows. It is not only important for her to be able to retrieve contact information for a particular person, but she also wants to get a better sense for groups of people in her network. She may need to look up all people in her research group if she has a question, or she may want to see all the people who live in a city she is about to visit.

1.1.2 Data-Specific Applications

Examples of "data-specific" applications include Apple's Address Book for managing contact information, iTunes[8] for organizing digital music files, and Picasa[15] for displaying photos. All of these applications are designed specifically for a type of data (ie. Person/Contact), assume that users care about a specific set of attributes or properties that define the data item (ie. Name, Address, Phone, etc), and define

[1]In this thesis, the term "type" in association with information or data is used to describe any set of data items that share a common "schema" or set of attributes or properties.

how users want to visualize and browse the data items (ie. List view). Moreover, these applications assume that all users organize and manage their data the same way. This can be powerful if they offer exactly what a user wants and needs, but otherwise can be quite limiting. In either case, users are dependent on the third party application developers for deciding how they want to manage their data.

Sally tries to use Apple's Address Book to manage these contacts. Address Book is a CRUD[10] application that allows users to Create, Read/browse, Update, and Delete people's contact information, including phone numbers, email addresses, and physical addresses. Its user interface reflects that of a traditional physical address book; the application sorts all the Person items alphabetically and users can categorize their contacts by manually creating "directories" and adding Person entries to each directory. It also has a search field that incrementally searches all text stored in the application. While this application offers the basic features of a CRUD application, Sally is limited in how she can visualize the entire collection, how she can interact with the collection, and exactly what item properties she can save. If Sally is going to a conference in the Bay area and wants to view all contacts who live in that area whom she can visit, the best she can do is manually create a directory for all Bay area contacts. She cannot view her contacts on a map based on home location and zoom into the Bay area, nor can she use faceted search to first filter contacts to all those who live in California, followed by those who live in any of the cities in the area. In addition, if someone moves, she must manually change that person's address as well as manually add or remove that person from any directory that may be based on home location.

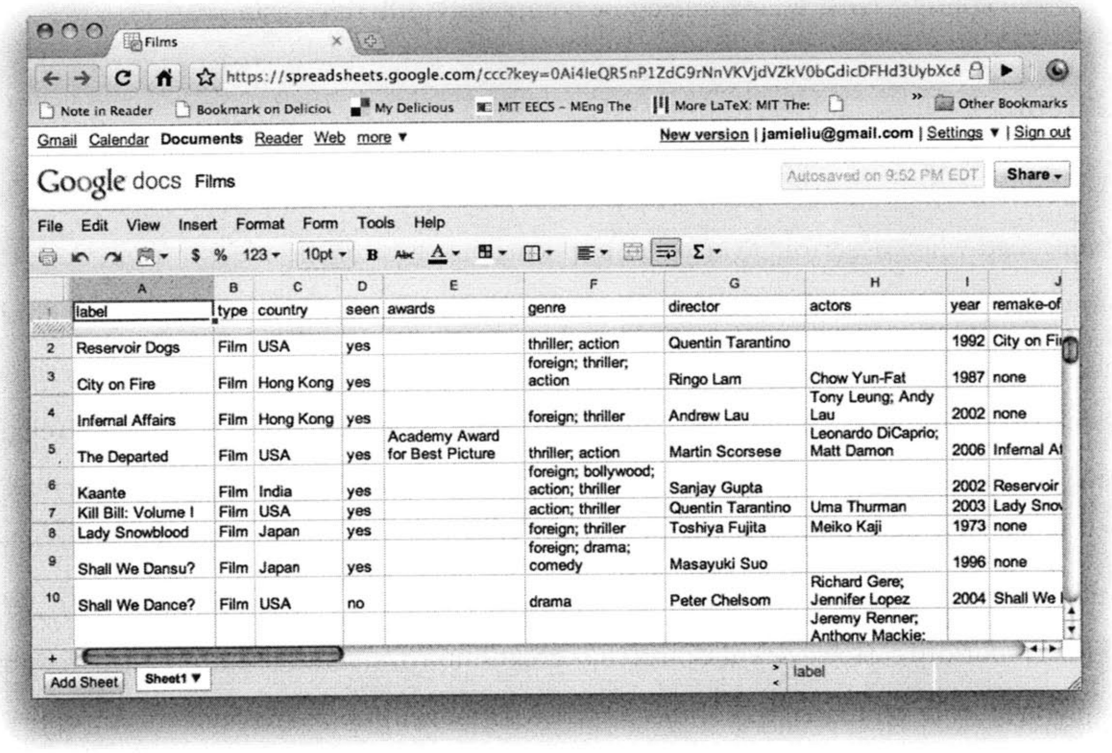
Another drawback in using an inflexible third party application is that while she is able to add custom labels for fields, all field types are for names, phone numbers, email addresses, or physical addresses, with the exception of a catch-all "Notes" field. If Sally wanted to add additional information about a person, such as "profession," "hobbies," or "papers-written," she has to add it to the catch-all "Notes" field. As a result, she cannot sort or filter on these custom properties. With this lack of custom support, Sally is severely limited in how she can interact with her data.

1.1.3 Generic Data Applications

Generic data applications, such as Excel[12] or Google Docs[4], are what we consider to be the second class of information management applications. They resolve the issue of allowing users to define the properties of their data, such that Sally can specify the "profession," "hobbies," "papers-written" and any other specific details she finds relevant. In addition, she can use such applications to manage more obscure data, where a specialized solution does not already exist. For example, she can manage information about "Trips" taken or "Inventory." However, despite supporting data customization, these applications still have their drawbacks. We discuss four generic data applications and their limitations in this section.

Spreadsheets

Spreadsheet applications such as Microsoft Excel[12] and Google Docs[4] offer a simple interface for entering and organizing data in a table format (see Figure 1-1). With these applications, Sally can choose to specify what type of data to save and any custom fields (or column headers) she may want to store for each item. For the Contacts example, she can now save the additional properties, such as "profession" and "papers written." She can sort her data on any column when browsing, but she is still limited in visualizations and browsing capabilities. Spreadsheets were originally designed to handle numeric data, and thus offer rich analyses to view and handle those data types. As a result, aside from sorting and viewing numeric data in charts and graphs, Sally can only visualize her data in a tabular view. She cannot view her data with an entirely different visualization, such as a Map view, nor can she use faceted browsing to filter her list of items to those with certain property values. This can be very restricting for someone who is trying to work with a multi-faceted data set.


Films
https://spreadsheets.google.com/ccc?key=0Ai4leQR5nP1ZdG9rNnVKVjdVZkV0bGdicDFHd3UybXcê
Note in Reader
Bookmark on Deliciou
My Delicious
MIT EECS - MEng The
More LaTeX: MIT The:
Other Bookmarks
Gmail Calendar Documents Reader Web more ▾
New version | jamieliu@gmail.com | Settings ▾ | Sign out
Google docs Films
Autosaved on 9:52 PM EDT
Share
File Edit View Insert Format Form Tools Help
Add Sheet
Sheet1 ▾
label

	A	B	C	D	E	F	G	H	I	J
1	label	type	country	seen	awards	genre	director	actors	year	remake-of
2	Reservoir Dogs	Film	USA	yes		thriller; action	Quentin Tarantino		1992	City on Fi
3	City on Fire	Film	Hong Kong	yes		foreign; thriller; action	Ringo Lam	Chow Yun-Fat	1987	none
4	Infernal Affairs	Film	Hong Kong	yes		foreign; thriller	Andrew Lau	Tony Leung; Andy Lau	2002	none
5	The Departed	Film	USA	yes	Academy Award for Best Picture	thriller; action	Martin Scorsese	Leonardo DiCaprio; Matt Damon	2006	Infernal A
6	Kaante	Film	India	yes		foreign; bollywood; action; thriller	Sanjay Gupta		2002	Reservoir
7	Kill Bill: Volume I	Film	USA	yes		action; thriller	Quentin Tarantino	Uma Thurman	2003	Lady Sno
8	Lady Snowblood	Film	Japan	yes		foreign; thriller	Toshiya Fujita	Meiko Kaji	1973	none
9	Shall We Dansu?	Film	Japan	yes		foreign; drama; comedy	Masayuki Suo		1996	none
10	Shall We Dance?	Film	USA	no		drama	Peter Chelsom	Richard Gere; Jennifer Lopez	2004	Shall We
								Jeremy Renner; Anthony Mackie;		

Figure 1-1: Google Docs spreadsheet

Bento

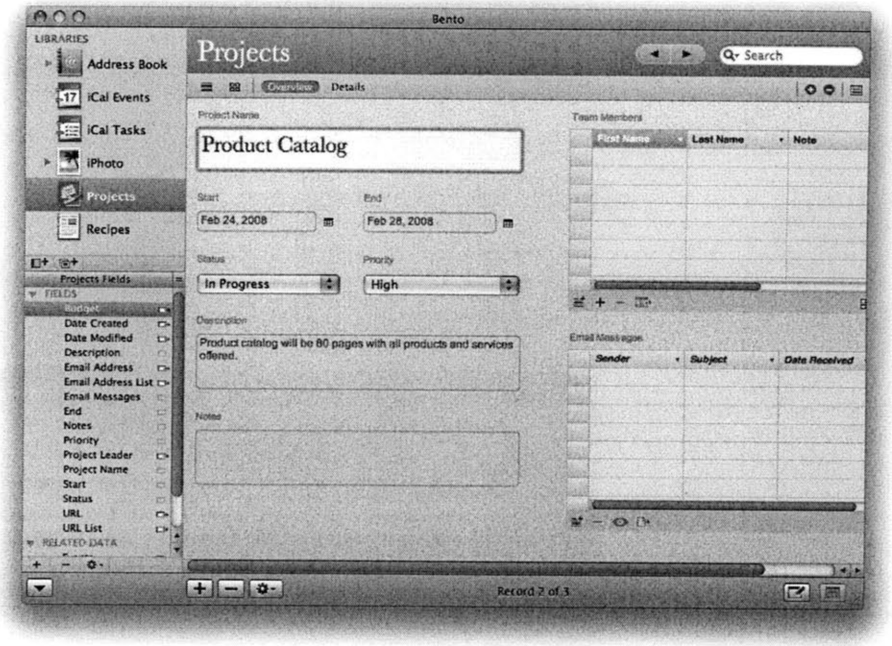
Filemaker, Inc.[3] offers a software application for Mac OS called Bento, which provides a few more advantages over spreadsheets in managing data, but remains limited in visualization and browsing options. Bento allows users to create a personal database without having to deal with configuring tables, designing a schema, and issuing query calls to retrieve data. Instead, users are given a nice graphical user interface to add, view, modify, and delete items. Unlike spreadsheets, Bento allows users to customize an input form by specifying what fields a given item should have (see Figure 1-2(a)). The input form supports certain media types, such as photos and video, rather than just text, and it can be viewed as either the main content or alongside the collection as a preview of the selected item (see Figure 1-2(b)). While spreadsheets offer a static table view, Bento provides that as well as a thumbnail view, shown in Figure 1-3. This, however, is the extent of how users can view their information. As with spreadsheets, a user cannot filter items by different facets of the information nor view the content with a customized or completely different visualization.

Exhibit

A publishing framework called Exhibit[7] helps overcome the visualization limitations mentioned when describing the other applications. Exhibit provides a framework that supports text search, faceted browsing, and several visualizations, including tabular, list, timeline and map views. Figure 1-4 shows a sample Exhibit of US presidents visualized on a map, with a search field and facets to filter the items based on religion and political party. This framework greatly reduces the barrier to entry for creating rich visualizations for a collection of information. Rather than designing a database and writing server-side scripts to retrieve information, authors only need to write HTML markup. However, despite this great improvement, users are still required to look at code and write the lines of HTML markup, which is not an appropriate solution for the layman end user.

Dido

While the official release of Exhibit did not support data editing, additional work has been done to add that capability to Exhibit such that users can now add new entries and modify existing ones. The resulting application is known as Dido (Data-Interactive DOcument)[9] and can be seen in Figure 1-5. Dido is a lightweight data management application and editor in a single HTML file. It offers a WYSIWYG editor for modifying the UI of an Exhibit, such that a user never needs to look at source code. One drawback, however, is that Dido stores all the data in the file itself. While this is beneficial in that all information is contained in a single file, it has its limitations. Creating multiple visualizations from the same data set entails duplicating data, and thus maintaining that data becomes error-prone as modifications for a single item will need to manually be made in multiple places. Overall, this work makes for a good foundation for the features desired in a customizable information manager, but more work needs to be done to support all the goals of this thesis. This will be discussed in more detail in Chapter 2.



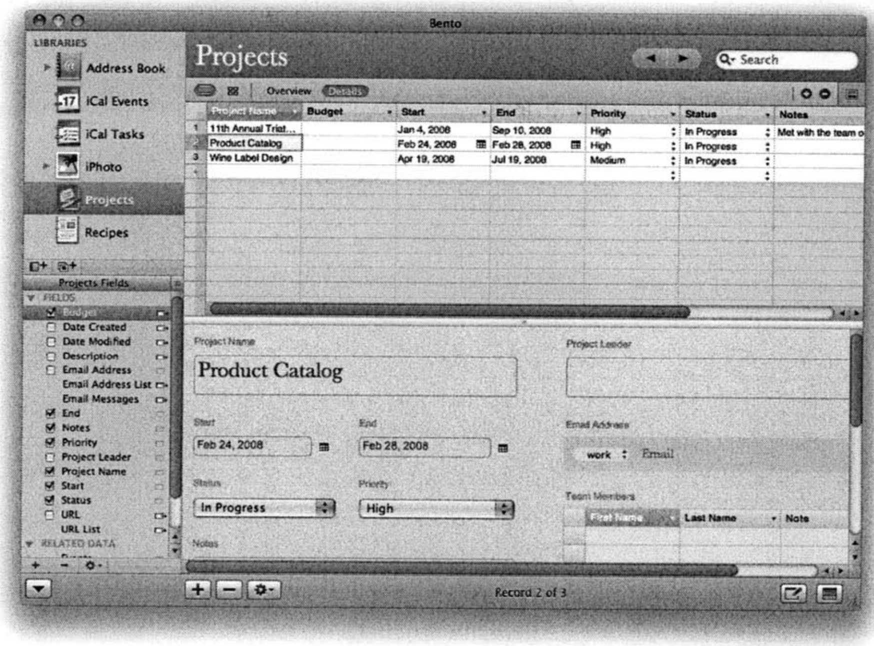
Project Name: Product Catalog

Start: Feb 24, 2008 End: Feb 28, 2008

Status: In Progress Priority: High

Description: Product catalog will be 80 pages with all products and services offered.

(a) Bento - Input form



	Project Name	Budget	Start	End	Priority	Status	Notes
1	11th Annual Triat...		Jan 4, 2008	Sep 10, 2008	High	In Progress	Met with the team o
2	Product Catalog		Feb 24, 2008	Feb 28, 2008	High	In Progress	
3	Wine Label Design		Apr 19, 2008	Jul 19, 2008	Medium	In Progress	

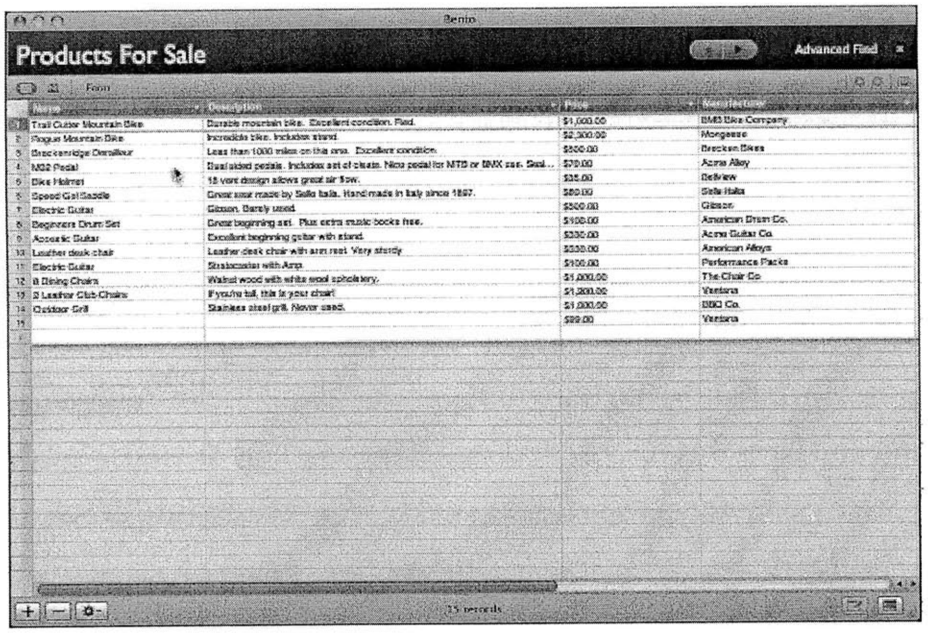
Project Name: Product Catalog

Start: Feb 24, 2008 End: Feb 28, 2008

Status: In Progress Priority: High

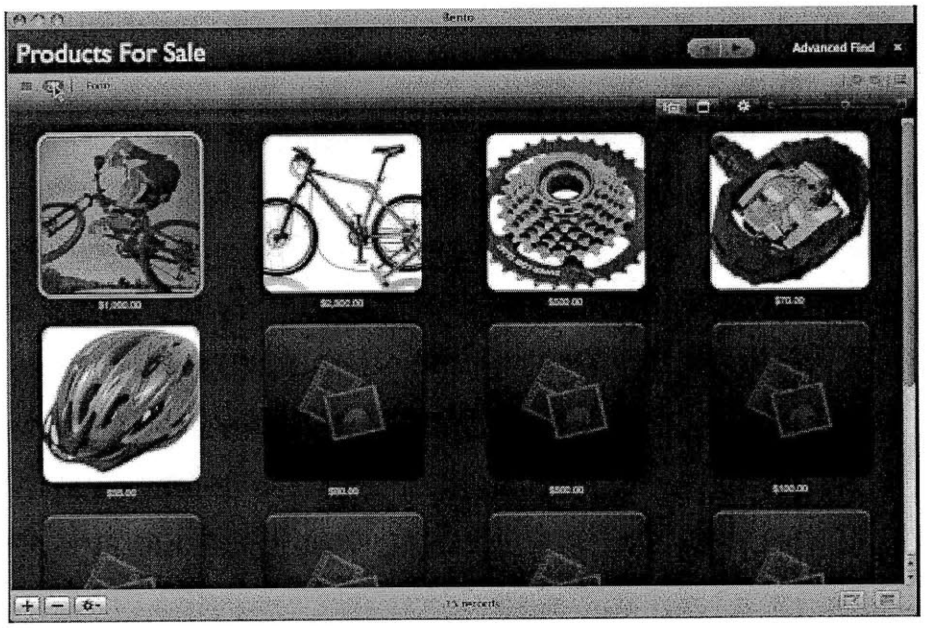
(b) Bento - Preview

Figure 1-2: Bento's item input form



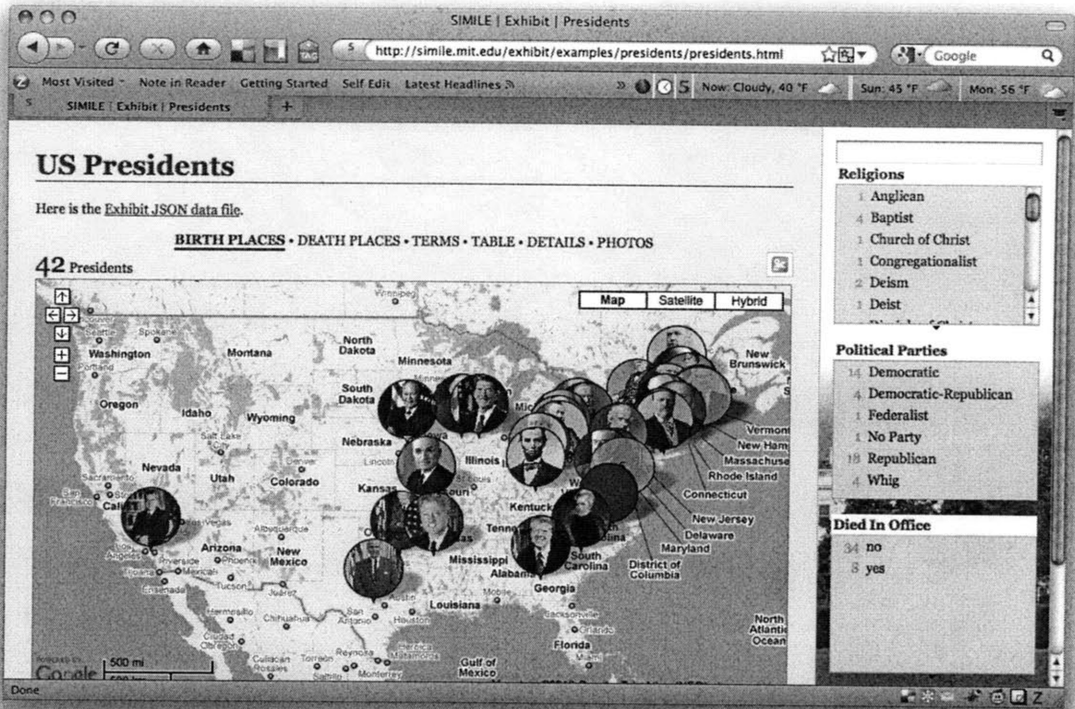
	Name	Description	Price	Manufacturer
1	Trail Cutter Mountain Bike	Durable mountain bike. Excellent condition. Red.	$1,000.00	BMS Bike Company
2	Rogue Mountain Bike	Incredible bike. Includes stand.	$2,300.00	Mongoose
3	Breckenridge Derailleur	Less than 1000 miles on this one. Excellent condition.	$500.00	Brecken Bikes
4	M32 Pedal	Dual sided pedals. Includes set of cleats. Nice pedal for MTB or BMX use. Seal...	$70.00	Acme Alloy
5	Bike Helmet	18 vent design allows great air flow.	$35.00	Bellview
6	Speed Gel Saddle	Great seat made by Sella Italia. Handmade in Italy since 1897.	$80.00	Sella Italia
7	Electric Guitar	Gibson. Barely used.	$500.00	Gibson
8	Beginners Drum Set	Great beginning set. Plus extra music books free.	$100.00	American Drum Co.
9	Acoustic Guitar	Excellent beginning guitar with stand.	$230.00	Acme Guitar Co.
10	Leather desk chair	Leather desk chair with arm rest. Very sturdy.	$330.00	American Alloys
11	Electric Guitar	Stratocaster with Amp.	$100.00	Performance Packs
12	8 Dining Chairs	Walnut wood with white wool upholstery.	$1,000.00	The Chair Co
13	2 Leather Club Chairs	If you're tall, this is your chair!	$1,300.00	Ventura
14	Outdoor Grill	Stainless steel grill. Never used.	$1,000.00	BBQ Co.
15			$99.00	Ventura

(a) List view



(b) Thumbnail view

Figure 1-3: Bento view options


SIMILE | Exhibit | Presidents
http://simile.mit.edu/exhibit/examples/presidents/presidents.html
Most Visited
Note in Reader
Getting Started
Self Edit
Latest Headlines
Map
Satellite
Hybrid
Done

US Presidents

Here is the Exhibit JSON data file.

BIRTH PLACES • DEATH PLACES • TERMS • TABLE • DETAILS • PHOTOS

42 Presidents

Religions

1	Anglican
4	Baptist
1	Church of Christ
1	Congregationalist
2	Deism
1	Deist

Political Parties

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

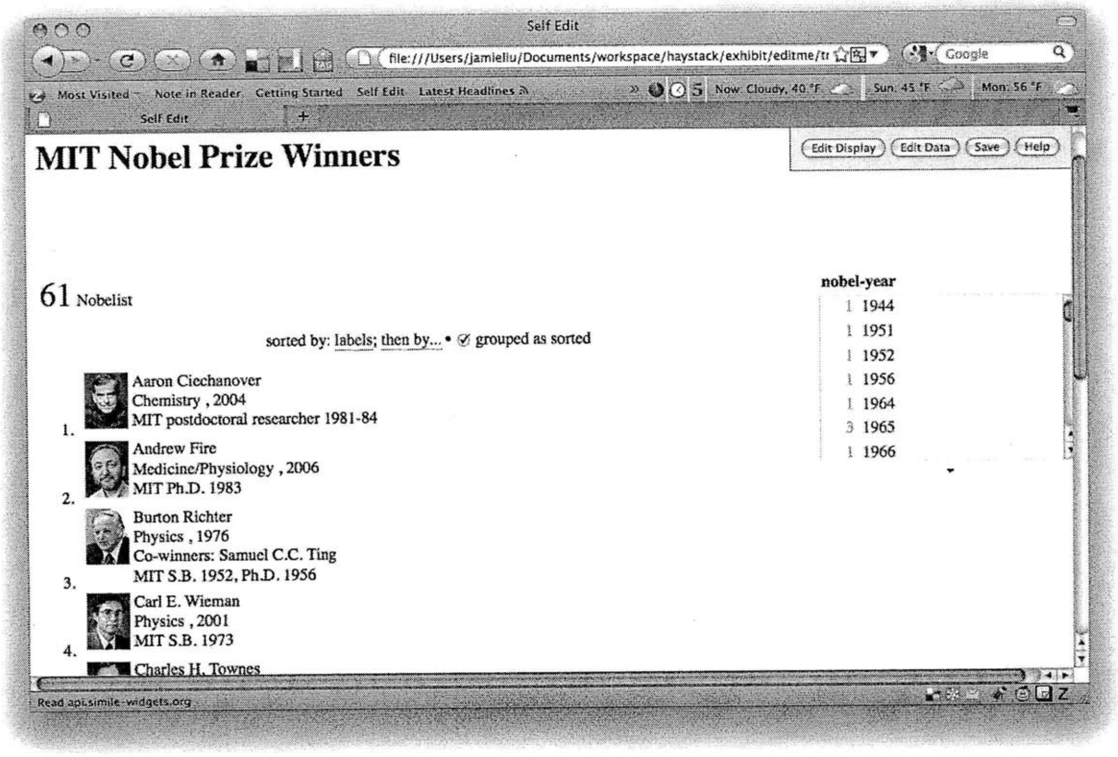
Died In Office

34	no
8	yes

Figure 1-4: Exhibit of US Presidents

1.1.4 Summary

Data-specific solutions are restrictive in how users can describe and browse their data, while generic data tools such as spreadsheets and Filemaker Inc.'s Bento are limited in how users can visualize and interact with their data. On the other hand, other applications, such as Exhibit and Dido, offer rich visualizations and browsing capabilities, but are not appropriate for the layman end user. Further, all these applications force a user to separate her data into different locations. Decentralized data becomes a big limitation when a user wants to visualize the same set of data or overlapping sets of data in different ways for different purposes. With existing applications, the user must save the same data in multiple places, which can lead to errors and inconsistencies in the data. We discuss this in more detail in Chapter 2.



MIT Nobel Prize Winners

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

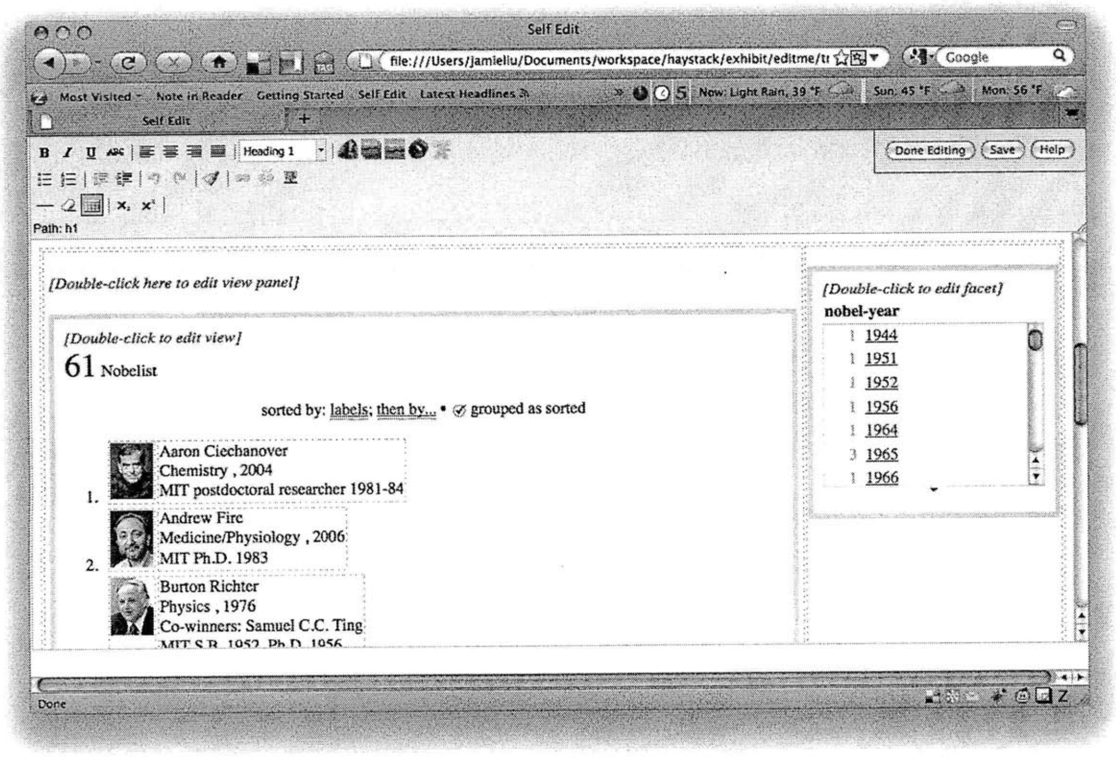
4. Carl E. Wieman
Physics , 2001
MIT S.B. 1973

Charles H. Townes

nobel-year

1	1944
1	1951
1	1952
1	1956
1	1964
3	1965
1	1966

(a) Dido - data manager



[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

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

[Double-click to edit facet]

nobel-year

1	1944
1	1951
1	1952
1	1956
1	1964
3	1965
1	1966

(b) Dido - WYSIWYG editor

Figure 1-5: Dido file

In summary, there are three main problems with these existing solutions:

- Users cannot define the data types or properties they find relevant
- Users are limited in how they visualize and browse their data.
- Users cannot easily visualize the same set of data or overlapping sets of data in different ways without saving the same data in multiple places.

1.2 Thesis Contributions

The research described here aims to provide an end-user application that addresses the aforementioned issues. It gives users the flexibility desired to customize the way they manage their personal collections of information. In particular, the features contributed in this thesis include:

- A local central data store for all items, allowing for multiple visualizations of the same data without risking data integrity.
- An interface for creating new visualizations and managing existing ones.
- A wizard and WYSIWYG editor for customizing the visualizations.
- Multiple ways to add or import data to the system, including adding new data from scratch and importing existing data from familiar formats used in spreadsheets.
- Multiple formats in which to export data to share with others, with and without the visualization.

With these features in mind, there are four different areas that garner attention: data storage, user interaction when creating a new visualization, user interaction when editing and interacting with the visualization, and the transfer of data to and from the application.

1.3 Thesis Outline

This thesis is divided into six chapters. The first chapter has outlined the motivations for this research and discusses existing applications that accomplish similar goals but remain as incomplete solutions. Chapter 2 describes existing tools that are leveraged and/or modified as part of the contributions made in this thesis project. Chapter 3 walks through the enhancements made to the user interface of Dido, one of the tools that is leveraged, to improve its usability. Chapter 4 describes the back-end design decisions made regarding the system architecture, as well as changes to the user interface that help accommodate performance issues. An evaluation of the system based on usability testing is presented in Chapter 5, and finally, Chapter 6 summarizes the contributions made in this thesis and identifies areas that can be pursued in future work.

Chapter 2

Background

Exhibit and Dido are powerful frameworks for creating rich and interactive websites on potentially dynamic sets of data. These tools are described in detail here, including benefits and limitations with respect to the goals and desired features of this thesis.

2.1 Exhibit

Exhibit[7] is a Javascript framework that makes it easy for website authors to create a dynamic, rich browsing experience for their end users without having to write any server-side code. With only a few lines of HTML, website authors can quickly create an interactive website that allows users to search, sort, or use faceted browsing to view and interact with data on the site.

Traditionally, if a website author wanted to create a web page that displayed some set of data, they had a few options based on their skill set. With only knowledge of HTML markup, they could create a static webpage that included the data along with the presentation. This was simple for the author to create, but gave end users a very limited viewing experience since they could only interact with a static list. This becomes unusable as the data set increases in size. To provide a more effective interface that allows users to sort, filter, and search the data, the website authors need to install and configure a database, add server-side scripting to issue queries to the database to retrieve the appropriate data, and create templates to specify how that data should be presented. With Exhibit, website authors only need to write a few lines of HTML and create a data source file that contains the structured data in a readable format, such as JSON[2]. This greatly decreases the barrier to entry for creating active websites, and authors are able to focus on what user interface (UI) components to include rather than how to support the backend.

The Exhibit data model is based on an array of "items," each of which has a set of "properties" that define the relationship between the item and the property value. Property values can be primitive data types, such as strings, dates, and URLs, but they can also be arrays of values or refer to other items in the database. Each item must have four properties: a *label* that is used to represent the item in a readable format, an *id* that uniquely defines the item, a *uri* for referring to the item in RDF[11] format, and a *type* for categorizing the item. All items of the same type share the same set of properties. Comparing this data model to that of a traditional relational database, each "item" can be seen as a database record, each "type" can represent a table, and the "properties" correlate to column or field names in a given table's schema.

Exhibit uses specific terminology for each UI component, primarily "views," "lenses," and "facets." Views are the visualization for the entire collection. The type of view used determines how all the items in the data set are laid out. For example, a Map view displays each item on a Google map (Figure 2-1), while a Timeline view places each item on a timeline (Figure 2-2). Simply adding a few lines of HTML markup with the attribute `ex:role="view"` allows the author to specify what type of view to add. For example, the following block adds the markup needed to create the map view seen in Figure 2-1.

```
<div ex:role="view" ex:viewClass="Map"
        ex:latlng=".latlng"
        ex:icon=".thumbnail"
        ex:shapeWidth="60"
        ex:shapeHeight="60">
</div>
```

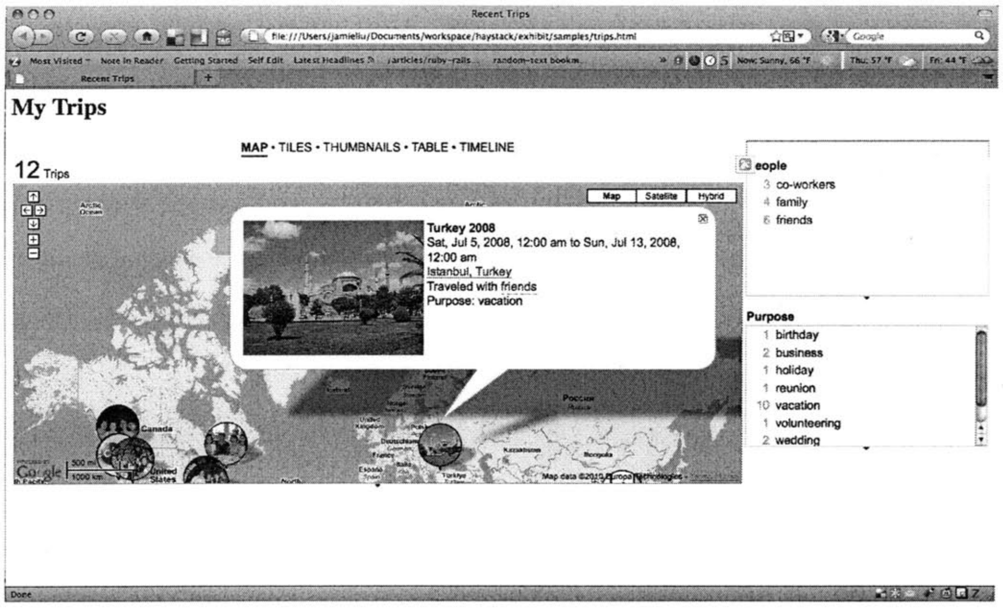


Figure 2-1: Exhibit Map view

Lenses, defined with the attribute `ex:role="lens"`, describe how each item in the collection is rendered. It defines the layout of the item's properties. Each property can be added with the custom `ex:content` tag. For example, an item's label can be added to the lens with the following markup:

```
<b><span ex:content=".label" class="name"></span></b>
```

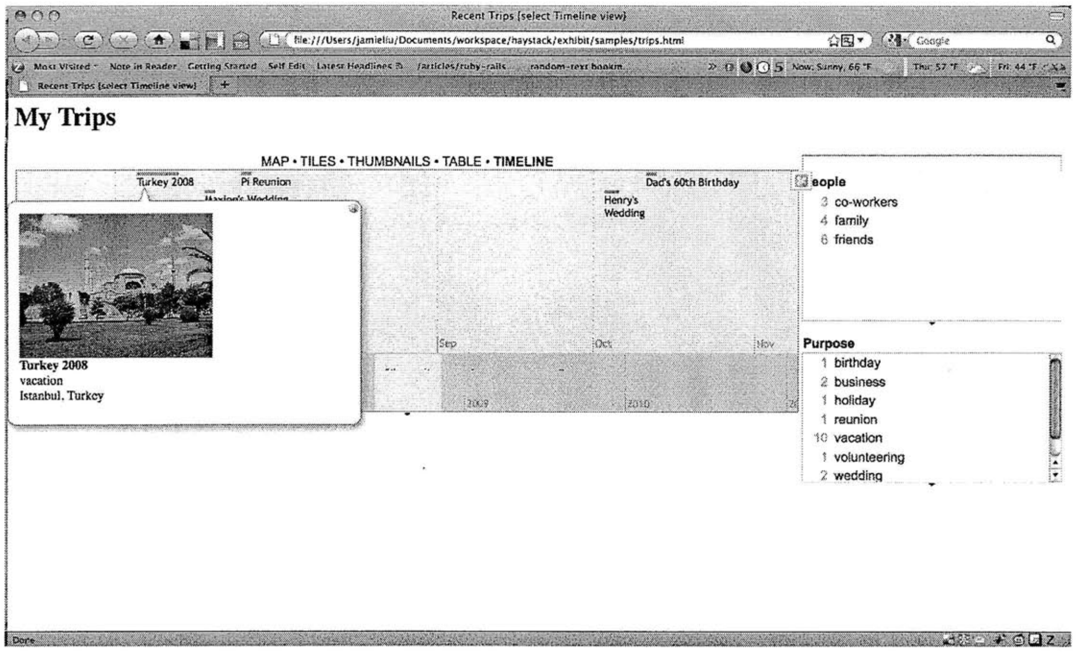
In Figure 2-3, this corresponds to the bolded "Turkey 2008" text.

Links and images can also be added to the lens. The underlined "Istanbul, Turkey" link in Figure 2-3 can be added with the `ex:href-content` attribute as follows:

```
<a ex:href-content=".location_link">
        <span ex:content=".location" class="location"></span>
</a>
```

while an image can be added with the `ex:src-content` attribute:

```
<div ex:if-exists=".thumbnail" class="thumbnail">
        <img ex:src-content=".thumbnail" />
</div>
```

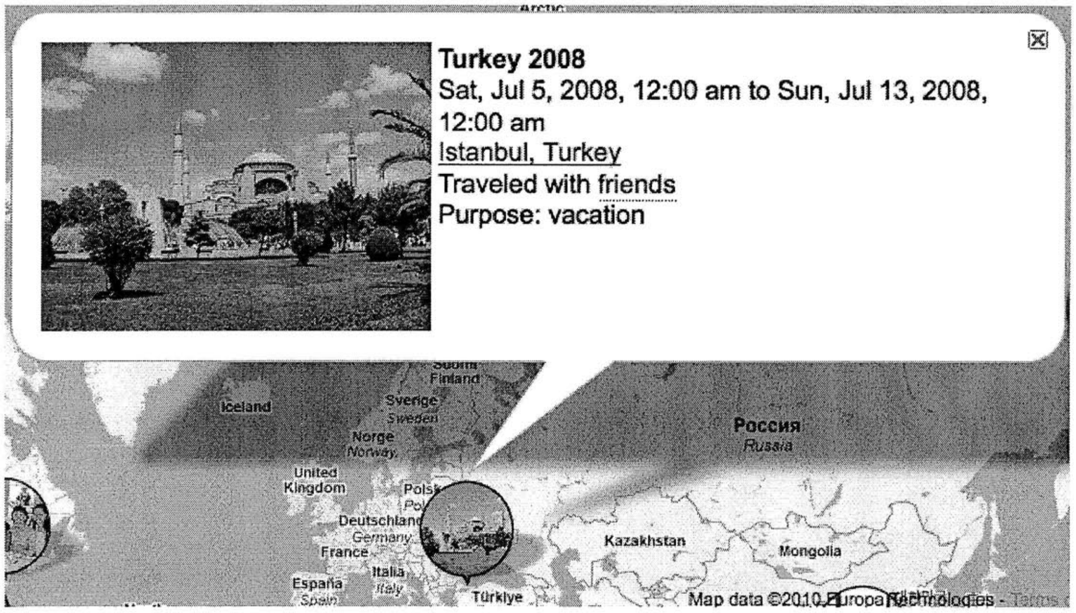

Recent Trips (select Timeline view)
file:///Users/jamieliu/Documents/workspace/haystack/exhibit/samples/trips.html
My Trips
MAP • TILES • THUMBNAILS • TABLE • TIMELINE
Turkey 2008
Pi Reunion
Dad's 60th Birthday
Henry's Wedding
Sep
Oct
Nov
2009
2010
People
Purpose
Done

Turkey 2008
vacation
Istanbul, Turkey

3	co-workers
4	family
6	friends

1	birthday
2	business
1	holiday
1	reunion
10	vacation
1	volunteering
2	wedding

Figure 2-2: Exhibit Timeline view



Turkey 2008
Sat, Jul 5, 2008, 12:00 am to Sun, Jul 13, 2008, 12:00 am
Istanbul, Turkey
Traveled with friends
Purpose: vacation

Figure 2-3: Exhibit lens used in Figure 2-1

Finally, the last major UI component in an Exhibit is the facet. Facets are filter widgets that allow a user to search or filter on any item property in their collection. There are several different types of facets to handle different ways a user may want to filter on a particular property of their data. Popular examples include a Text Search facet, which adds an incremental search field to the display, and a List facet, which allows users to filter their collection to show only items with the selected values. For example, Figure 2-4 filters the 12 trips down to only 6 involving "friends."

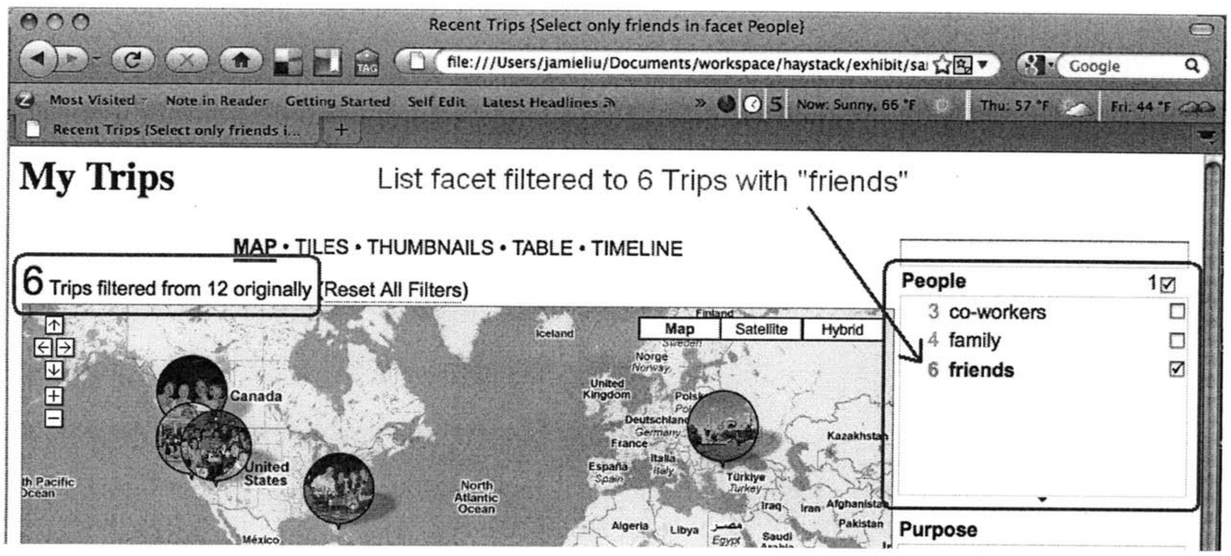


Figure 2-4: List facet for "People" in use

For further customization, the author can control the layout and style of any of the components by using HTML and CSS. This gives website authors much expressive power to create highly customized interactive websites without any server-side scripting.

Aside from the user interface components, website authors also have a lot of flexibility in how they organize their data. They decide how to separate items into different data types, including what properties each type of item can have. Since the data is stored in a format inspired by the Web Consortium's Resource Description Framework (RDF) standard [11], the website authors can also easily change the schema they are using for each type (ie. add a new property or remove an existing one) at any time, without making any changes to Exhibit's underlying database structure.

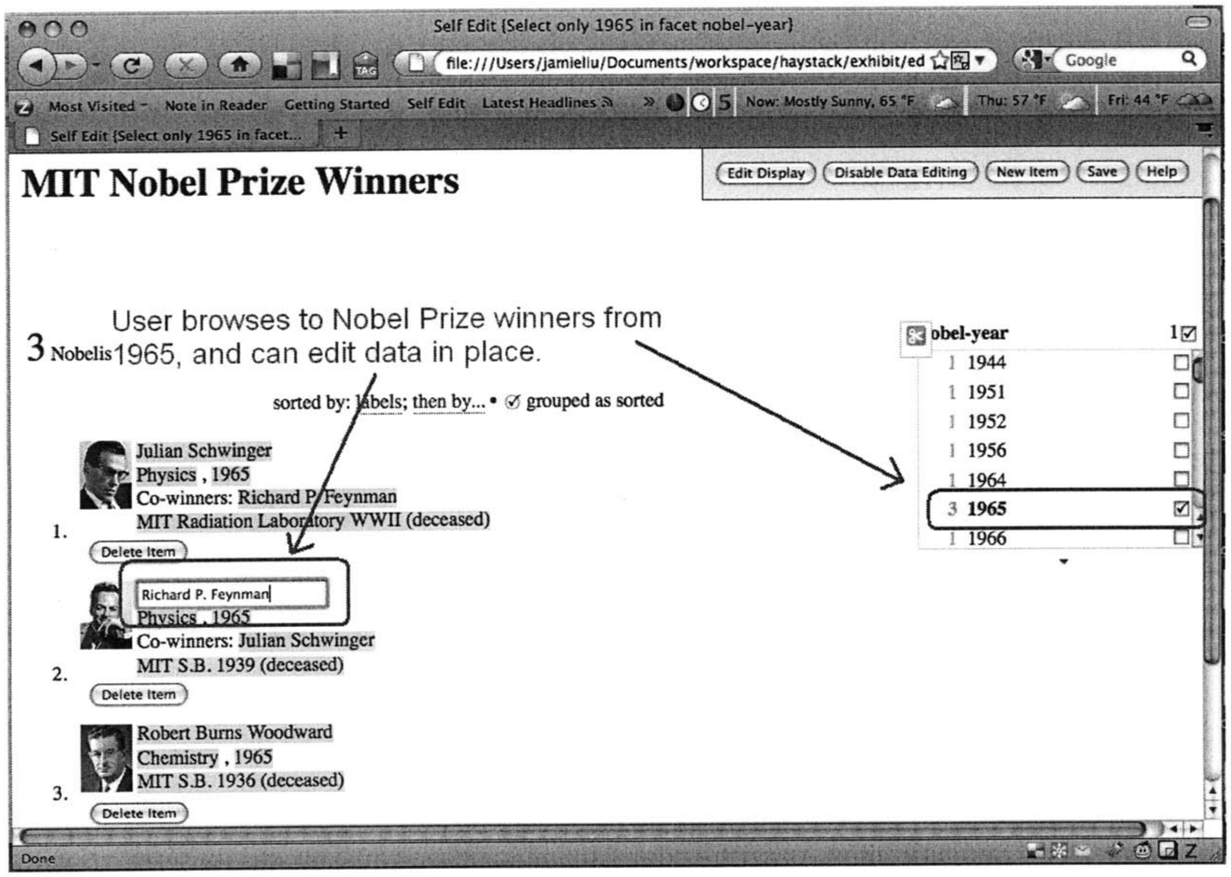
By offering a highly customizable visualization and rich browsing experience, Exhibit provides a good foundation for allowing application authors to create customized visualizations of their information. However, it still has limitations as an information manager for end users. while Exhibit decreases the barrier to entry by not requiring authors to do any database programming or server-side scripting, it still requires authors to write lines of markup using Exhibit's custom HTML attributes to specify what type of views to use, what properties to be displayed, etc. Asking a casual end-user to specify this information in the markup language is not reasonable. In addition, Exhibit was created to work with a static set of data. Once that data was loaded, users could interact with that set of information but could not modify or add new items to the collection. Support is needed for both of these issues such that an end user like Graduate Student Sally can customize her visualization and browsing preferences, as well as edit her data, without writing any code.

2.2 Dido

Dido addresses Exhibit's limitations by providing a WYSIWYG editor for modifying the Exhibit's visualization, as well as offering a data editing mode to allow users to modify the Exhibit's data[14]. Dido is a lightweight data management application and application editor in a single HTML file. It offers many useful features, including the aforementioned editing features, but it also has its limitations. These benefits and drawbacks are described in this section.

2.2.1 Overview

Dido provides capabilities for an end user to visualize and edit their data, as well as modify any part of the overall visualization, without writing any code. Using Exhibit's framework, Dido allows users to browse and interact with data as they would with any Exhibit. By using the data editing enhancements added to Exhibit, Dido is able to provide an interface that allows users to modify the data contents in the Exhibit while viewing and interacting with it. For example, if a user is filtering the data and notices that an item's contents need to be updated, the user can simply turn on the data editing mode and modify the item in place. He or she does not need to switch to an entirely different editing view that resets the items currently on display in order to edit the data. For example, as seen in Figure 2-5, a user can edit data in place after filtering contents to show only Nobel Prize winners from 1965.



MIT Nobel Prize Winners

User browses to Nobel Prize winners from 1965, and can edit data in place.

3 Nobelis

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

1. Julian Schwinger
Physics , 1965
Co-winners: Richard P Feynman
MIT Radiation Laboratory WWII (deceased)

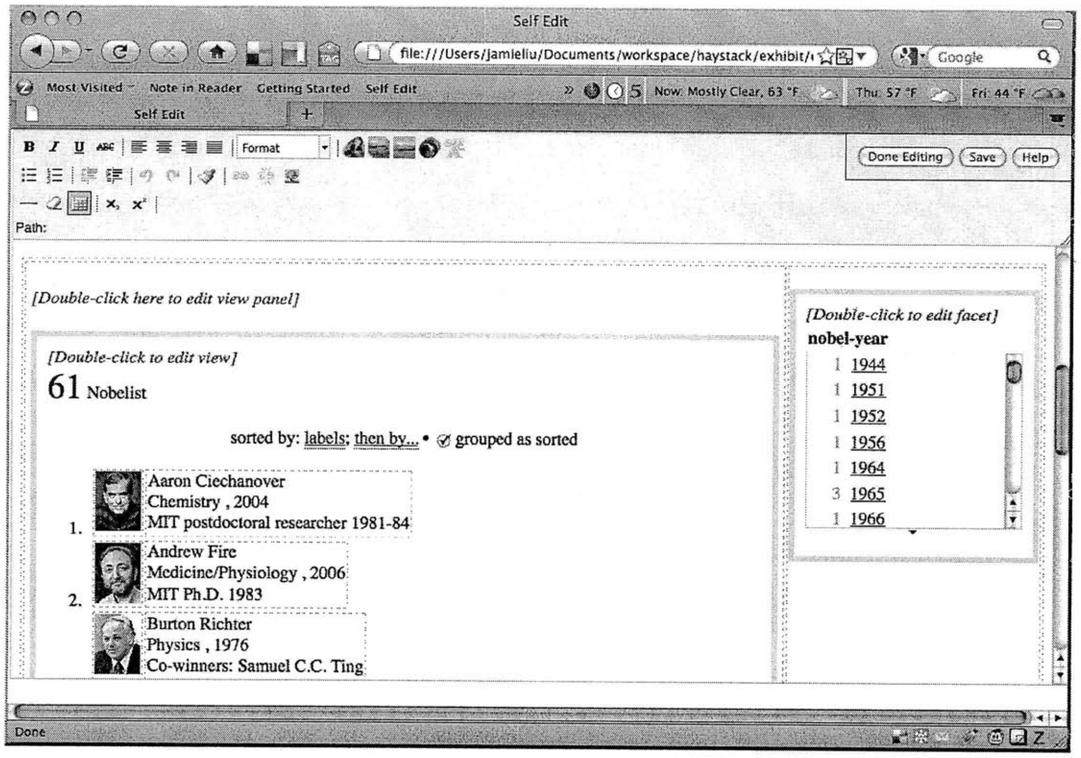
2. Richard P. Feynman
Physics , 1965
Co-winners: Julian Schwinger
MIT S.B. 1939 (deceased)

3. Robert Burns Woodward
Chemistry , 1965
MIT S.B. 1936 (deceased)

obel-year	1 ☑
1 1944	☐
1 1951	☐
1 1952	☐
1 1956	☐
1 1964	☐
3 1965	☑
1 1966	☐

Figure 2-5: User is changing Feynman's name while browsing Nobel prize winners from 1965.

In addition, Dido uses the tinyMCE[17] WYSIWYG editor to allow users to edit the visualization with which they are currently viewing and interacting (see Figure 2-6). Rather than write any code or markup to customize the application's display, a user can simply use the editor and immediately view the results of his or her changes.


Self Edit
file:///Users/jamieliu/Documents/workspace/haystack/exhibit/
Done Editing
Save
Help
Path:

[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

Burton Richter
Physics , 1976
Co-winners: Samuel C.C. Ting

[Double-click to edit facet]

nobel-year

1	1944
1	1951
1	1952
1	1956
1	1964
3	1965
1	1966

Figure 2-6: Dido's WYSIWYG editor

2.2.2 Limitations

While Dido provides an effective interface for creating a customizable data management application for the casual end-user, it also has its limitations. First, the data for a Dido exhibit is stored in the file itself. This decentralized data model can lead to the loss of data integrity, as mentioned in Chapter 1. If a user wanted to reuse the same set of data for multiple visualizations, he or she would need to copy the data over to a new Dido file. As with redundant information in any database[1], this duplicate data leads to an unnecessary use of resources to maintain the accuracy of the information. Modifying an item that is used in n number of visualizations requires n number of modifications. This can cause errors and inconsistencies in the data when updates are not made to all copies.

To demonstrate this with a more concrete example, let us use our Grad Student Sally persona from Chapter 1. Sally wants to maintain a collection of all people she knows, as well as a collection dedicated to researchers and publications in her field. Let us assume that the set of researchers whom she follows is a subset of the larger collection of people whom she knows, and that the properties relevant when viewing general contact information for all people is different from those relevant when viewing researchers in particular. For example, Sally may want to see the papers written by the researchers she pays attention to, but not those written by her friends and family. Using Dido files alone, she would need to do dual maintenance on those researchers, which increases the risk of data errors and inconsistencies. If she meets a new colleague at a conference, she must remember to add that colleague's information to both visualizations, possibly adding an inconsistency in what data items are present in each visualization. If a researcher changes his email address, Sally needs to manually update that information in two places. If she makes a typo in one, it will be ambiguous which email address is correct, adding an inconsistency within the item's properties themselves. This problem worsens for each visualization created for a given set of data.

Because of these limitations in Dido's data store, modifications were necessary to leverage Dido's editing capabilities while also utilizing a more centralized storage system. The resulting system would give users the flexibility of creating any number of visualizations for the same set of data while maintaining data integrity. In addition, changes were needed to make Dido more usable and to help new users get acquainted with the UI more easily. Support for importing and exporting data to be shared with others were important additions to the system, as well.

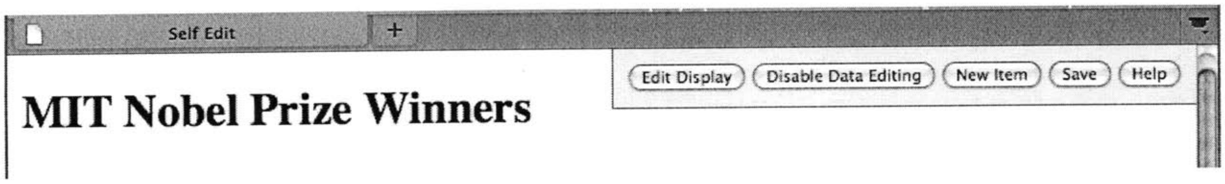
Chapter 3

Dido Enhancements

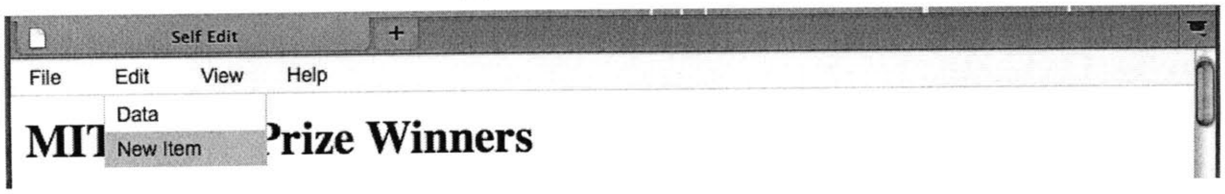
Dido provides the core functionality needed for a customizable end-user information management system, but it has some drawbacks with respect to usability. This section describes the enhancements made directly to Dido to help improve its usability and help users get started using it, while Chapter 4 describes features added to the system to support Dido files backed by a centralized data store.

3.1 Dido Controls

Standalone Dido originally had a control panel in the top right with a few buttons for the "Edit Display," "Edit Data," "Save," and "Help" actions (see Figure 3-1). When extending Dido to support a centralized data store (as described in Chapter 4), more actions were available and/or necessary to include. Adding more actions to the control panel would increase the amount of space it took up, add to the clutter of buttons, and make each specific action harder to find for the user. By switching from a control panel metaphor to a standard menu bar at the top of the page, we could add more actions without cluttering the interface and confusing a user. Adding a menu bar also increases the consistency of Dido with other desktop applications, thereby creating a more familiar interface for the user.



(a) Original Control Panel



(b) Menu Bar

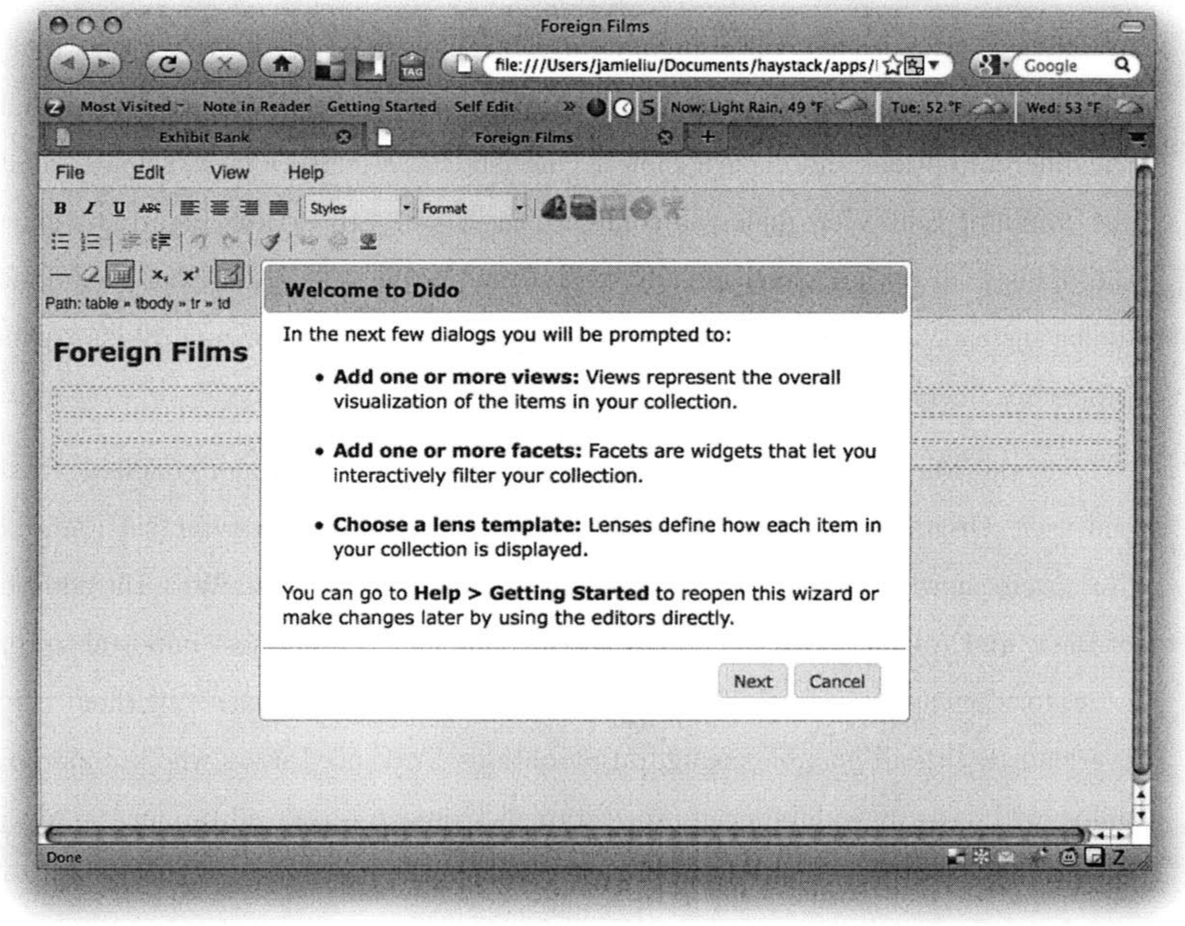
Figure 3-1: Control Panel vs Menu Bar UI

3.2 Wizard and Updated Dialogs

When first launching the editor mode of a Dido file, a new user may not know where to begin to create a custom visualization. In order to improve the learnability of this application, a Getting Started wizard was added to guide users, such as our persona Sally, through the different steps of creating an exhibit. If no visualization exists for the Dido file, the Editor view is opened and the wizard is launched immediately. The wizard takes Sally through the steps of creating a visualization from scratch. Once she is done with the wizard, she can be reopen the wizard by launching an action in the Help menu. If a visualization already exists when Sally launches the wizard, she has the option to keep the existing UI or overwrite/modify it.

Welcome

A welcome screen, shown in Figure 3-2, is the first dialog of the Getting Started wizard. It outlines the next steps to be shown in the wizard, and more importantly, introduces a new user to the different UI components that can be modified in the editors. This is especially important for users unfamiliar with Exhibit terminology.



Welcome to Dido

In the next few dialogs you will be prompted to:

- **Add one or more views:** Views represent the overall visualization of the items in your collection.
- **Add one or more facets:** Facets are widgets that let you interactively filter your collection.
- **Choose a lens template:** Lenses define how each item in your collection is displayed.

You can go to **Help > Getting Started** to reopen this wizard or make changes later by using the editors directly.

Figure 3-2: Getting Started wizard launched for new Dido

View Picker

An exhibit can have multiple views. A view panel is a container for views, and provides an interface to allow end users to switch between the views within the view panel. If a view panel does not already exist, a new one is created and the first step in the wizard asks the user to choose what type of view to use. Sally is a new user who is unfamiliar with Exhibit and does not know what types of view options are available to visualize her collection of items. Therefore, we added a view picker dialog with image and text descriptions for each type of view to help her better understand the different options and make a choice. Once she has selected a view, she is taken to the standard Edit View dialog to configure the view's properties. Figure 3-3 shows the dialogs. If a view panel already exists, the system opens the Edit View Panel dialog instead. This allows the user to either add a new view or modify view panel properties, including what the initial view should be.

The view dialogs were modified from the original Dido to be more readable for an end user. Originally, a list of all properties for a particular view type (see Figure 3-4(a)) were shown to the user at once, most of which were optional. With the addition of Basic and Advanced tabs (Figure 3-4(b)), Sally can see the most important properties for the basic case, including required properties that must be set, without being overloaded with all possible configuration options. Advanced users who are more familiar with creating exhibits can proceed to click on the Advanced tab for the rest of the available settings. In addition, for views where the user needs to specify a list of multiple expressions and properties for each expression (ie. ascending or descending sort order), we replaced the text fields that require manual input of expression values and properties with dropdown controls. This increases the efficiency with which users can enter values, and decreases the risk of errors arising when manually typing in values.

When Sally is done adding a view, a preview of it populates in the editor behind the wizard dialog (see Figure 3-5). The user is prompted to either add another view or continue by pressing "Next." Choosing the "Add another view" option will launch

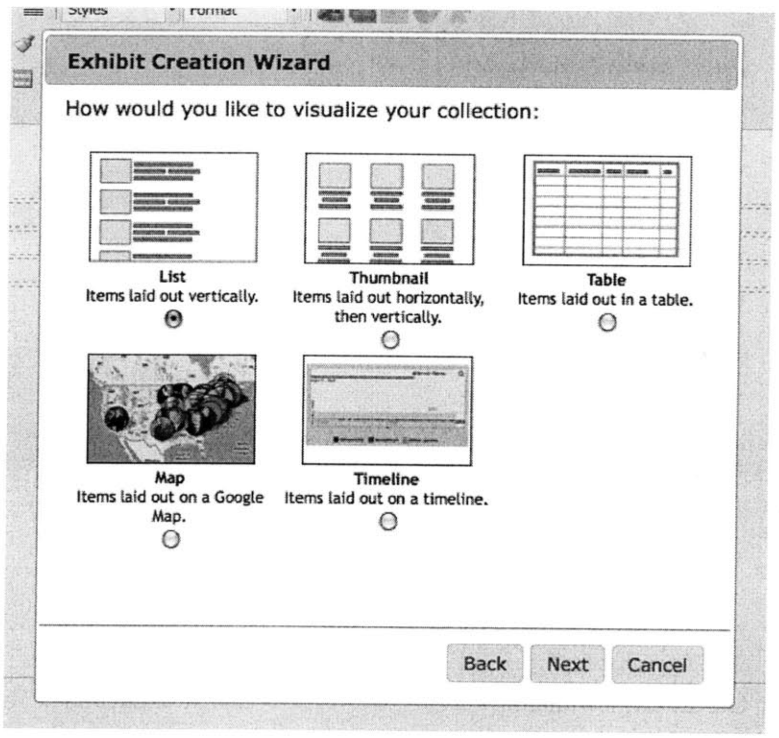
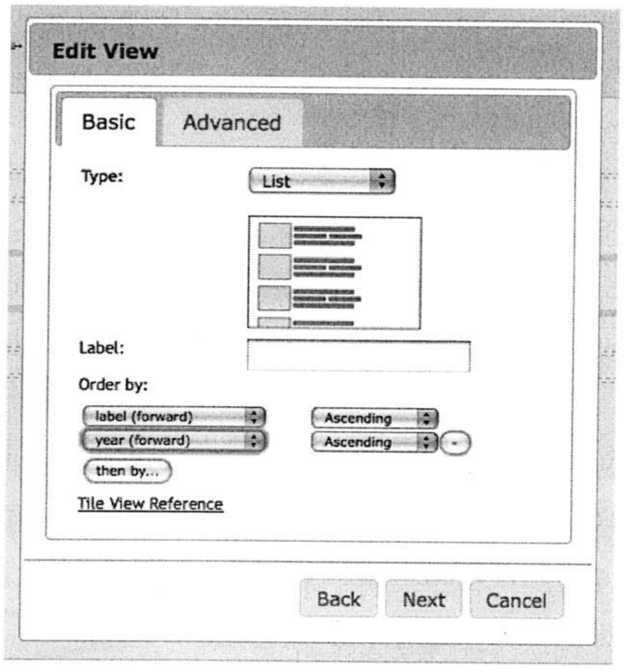


Exhibit Creation Wizard

How would you like to visualize your collection:

List Items laid out vertically. [✓]	**Thumbnail** Items laid out horizontally, then vertically.	**Table** Items laid out in a table.
Map Items laid out on a Google Map.	**Timeline** Items laid out on a timeline.	

(a) View Picker dialog



Edit View

Type: List

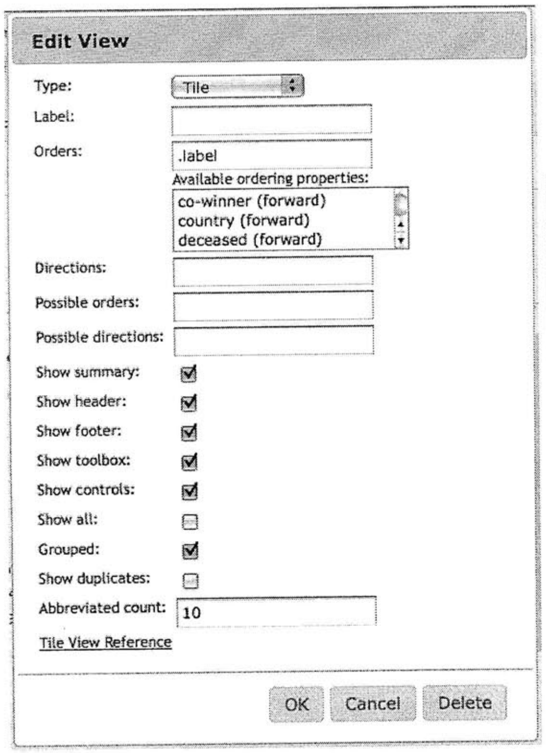
Label:

Order by:
label (forward) Ascending
year (forward) Ascending
then by...

Tile View Reference

(b) Edit View dialog with type preselected with selection from View Picker dialog.

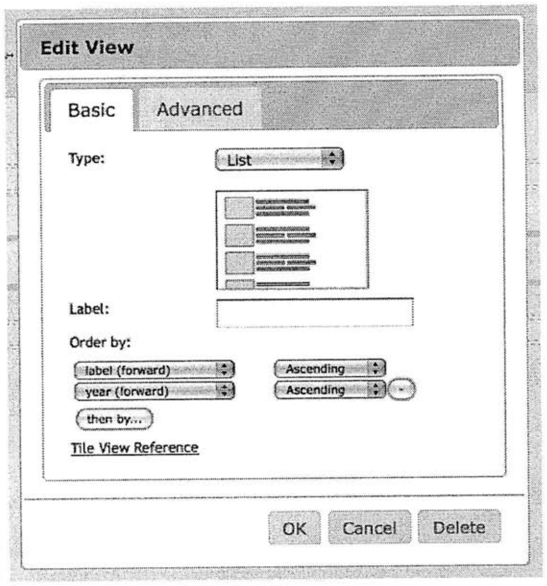
Figure 3-3: View configuration dialogs



Type:	Tile
Label:	
Orders:	.label
Available ordering properties:	co-winner (forward) country (forward) deceased (forward)
Directions:	
Possible orders:	
Possible directions:	
Show summary:	✓
Show header:	✓
Show footer:	✓
Show toolbox:	✓
Show controls:	✓
Show all:	☐
Grouped:	✓
Show duplicates:	☐
Abbreviated count:	10

Tile View Reference

(a) Old Edit View dialog



Type:	List	
Label:		
Order by:	label (forward)	Ascending
	year (forward)	Ascending

Tile View Reference

(b) New Edit View dialog

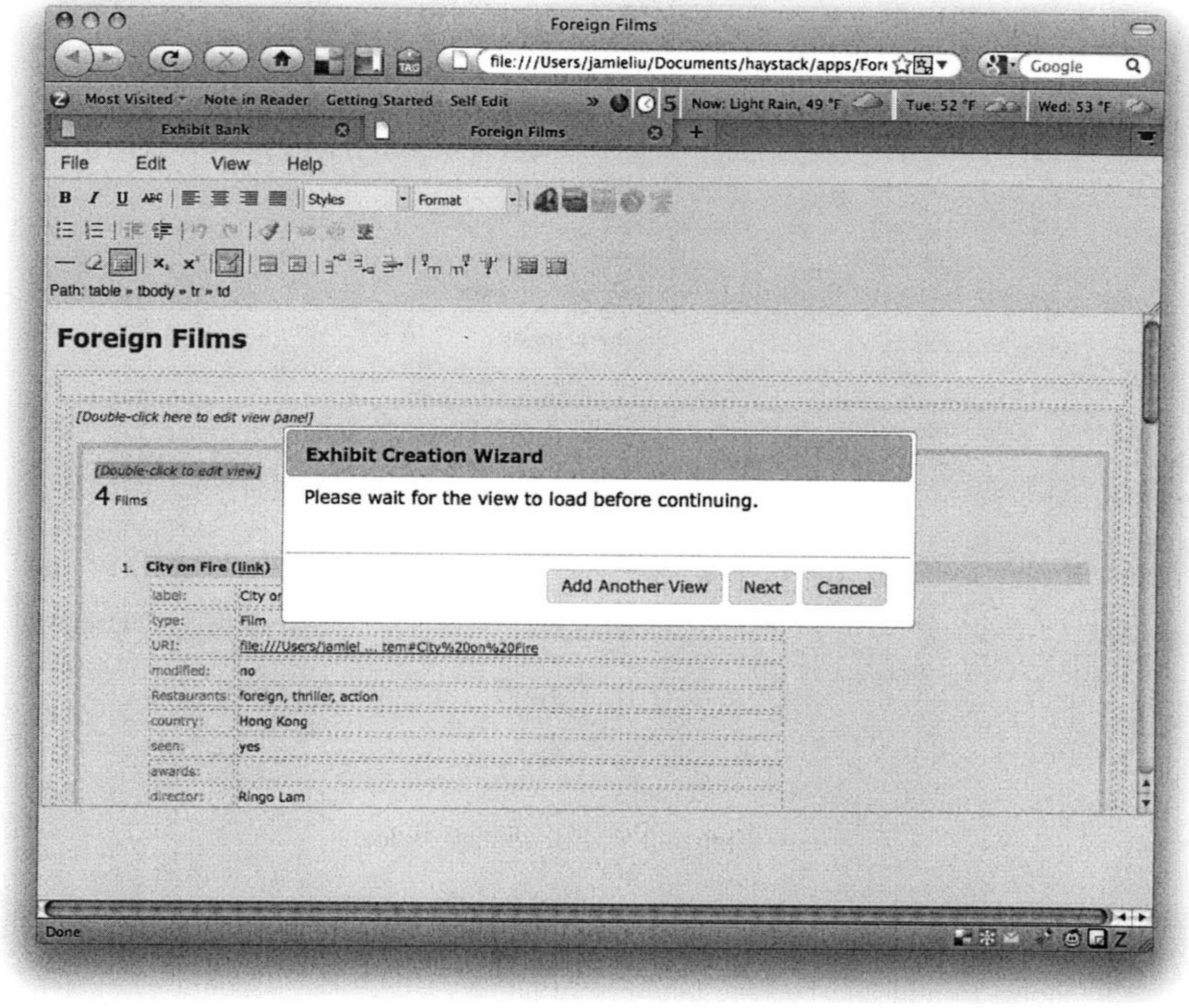
Figure 3-4: Edit View dialog changes

the view picker dialog again.

Facet Picker

If Sally chooses to continue, a location picker dialog is shown to allow her to choose where to put a facet, or filter widget, relative to the view: Right, Left, Top, and Bottom (see Figure 3-6). This gives users some customization abilities from within the wizard. We provide basic layout options that a new user may want to help them get started. Advanced users can further customize the layout of their visualizations by directly using the editors themselves.

After choosing where to add the facet, Sally is prompted with the Edit Facet dialog (see Figure 3-7). Here, she can specify what type of facet to add, with options including List, Text Search, and Range. Similar to the Basic and Advanced tabs added for the view dialogs, the same concept for tabs were applied to the Edit Facet dialogs. This allows her to quickly create a facet without getting caught up or confused



Foreign Films

[Double-click here to edit view panel]

[Double-click to edit view]

4 Films

1. City on Fire (link)

label:	City o
type:	Film
URI:	file:///Users/jamiel ... tem#City%20on%20Fire
modified:	no
Restaurants:	foreign, thriller, action
country:	Hong Kong
seen:	yes
awards:	
director:	Ringo Lam

Exhibit Creation Wizard

Please wait for the view to load before continuing.

Figure 3-5: Dialog after user has added a view, allowing user to add another one or continue.

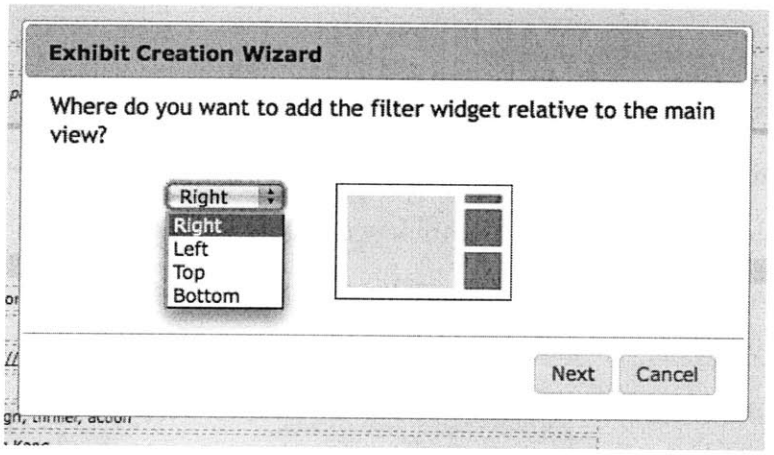


Exhibit Creation Wizard

Where do you want to add the filter widget relative to the main view?

Figure 3-6: Facet Location Picker dialog

looking at optional properties.

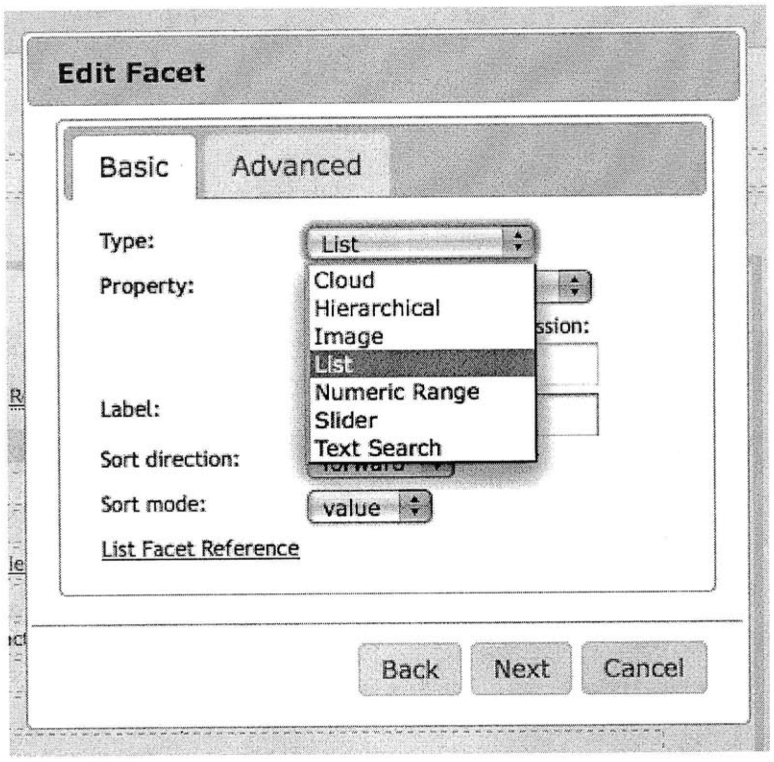


Figure 3-7: Edit Facet dialog

After adding the facet, Sally is prompted in the next step of the wizard with the choice to add another one before continuing. Under the assumption that users typically want to create more than one facet to help browse their data, this provides a convenient method to add multiple facets quickly.

Lens Picker

Finally, after Sally is done adding facets (and therefore done specifying UI components for the overall display), she is taken to the Lens editor to specify how each item should be rendered. New users unfamiliar with the lens editor may not know how to add lens content to their visualization. Among existing Exhibits[1], a few layouts are seen repeatedly. Under the assumption that users prefer similar layouts or can tweak such layouts to suit their needs, we created a lens template picker with a few basic layouts to help users get started. These options are shown in Figure 3-8. We give

[1]Some examples can be seen at http://simile.mit.edu/wiki/Exhibit/Examples.

users four options:

- **Existing or Blank:** Whether a new exhibit or not, this option allows users to continue without creating a new lens. For a new exhibit, this option leaves the lens editor empty and for an existing exhibit that already has a lens defined, this option keeps the existing lens(es).
- **Labels:** This lines up property labels and values side by side.
- **Landscape:** This option, common in Tile view layouts, allows users to specify an image on the left, and show property values on the right.
- **Portrait:** This option, common in Thumbnail view layouts, allows users to specify an image on top, and show property values below it.

If Sally selects either the landscape or portrait templates, she is given the option to choose a property that refers to the image she wants to display, as well as the ability to specify a width and/or height for the image (see Figure 3-9). Because a user may be gathering images from different resources, the images may vary in size. This offers a way to ensure that they are uniformly displayed, creating a neater, cleaner-looking visualization. If either of the size attributes are left blank, the default values for the image are used.

Once Sally selects a template and clicks Next, a lens is created dynamically based on the properties of the data items in the exhibit and it is added to the Lens Editor. Another assumption made when designing the dynamic lens picker is that it is easier for new users to modify or delete lens content already in the editor than it is to create lens content from scratch. Thus, we populate the lens with all properties available for the given types in the exhibit. This provides a starting point for the user, who can then add text and rearrange, delete, or otherwise modify any lens content while working with the actual Lens editor. Figure 3-10 shows the lens added to the editor behind the wizard dialog.

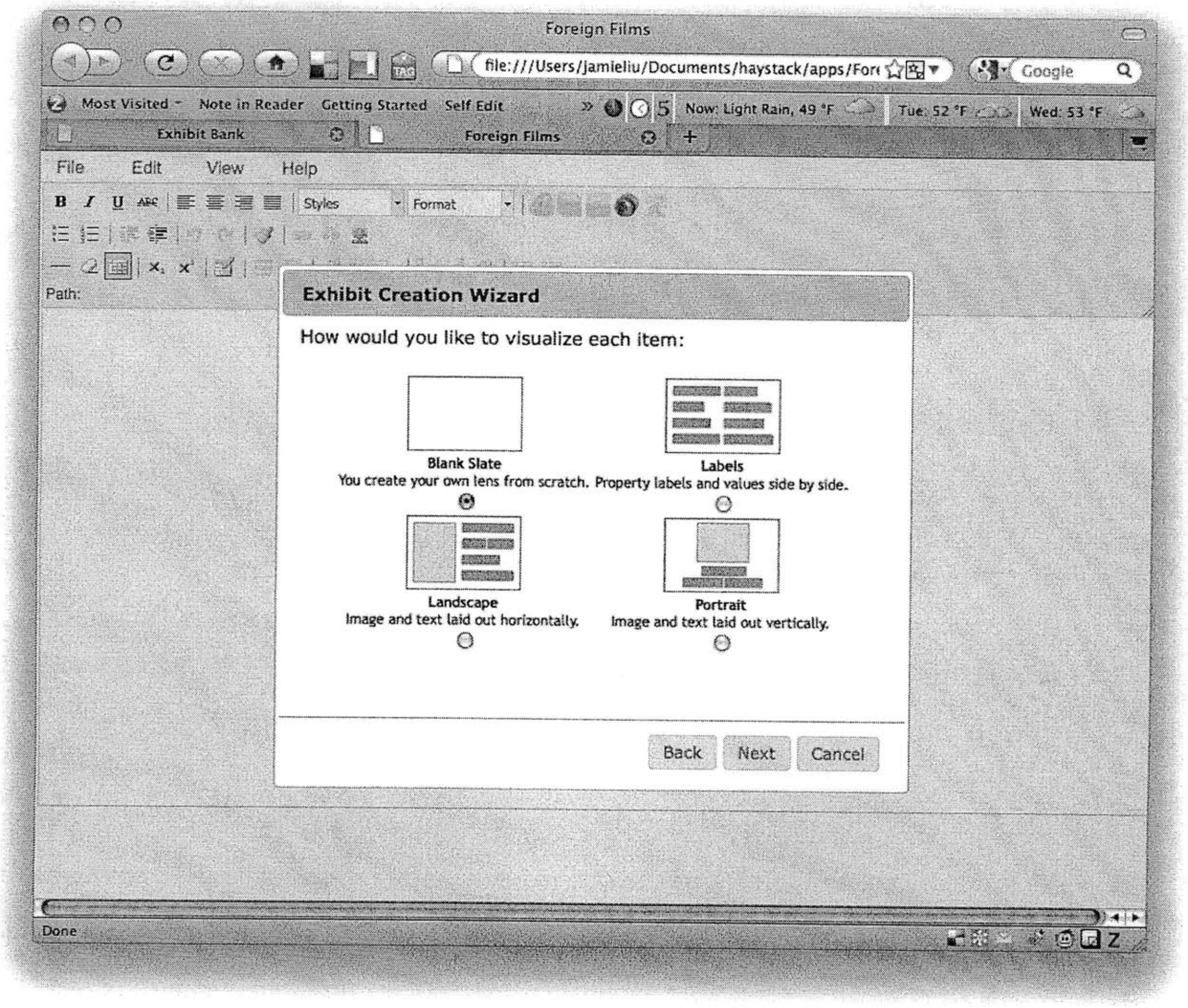


Exhibit Creation Wizard

How would you like to visualize each item:

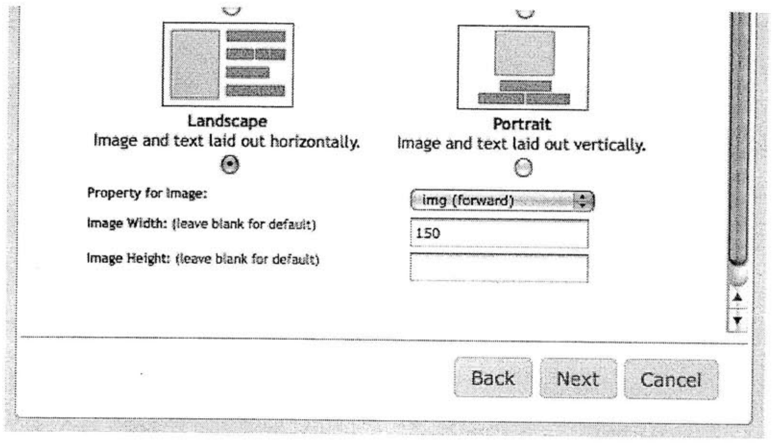
Blank Slate
You create your own lens from scratch. [✓]

Labels
Property labels and values side by side.

Landscape
Image and text laid out horizontally.

Portrait
Image and text laid out vertically.

Figure 3-8: Lens Template Picker, with empty lens editor behind the dialog.



Landscape
Image and text laid out horizontally. [✓]

Portrait
Image and text laid out vertically.

Property for Image:	img (forward)
Image Width: (leave blank for default)	150
Image Height: (leave blank for default)	

Figure 3-9: When landscape or portrait options are selected in the Lens Template Picker, the user can enter an image property.

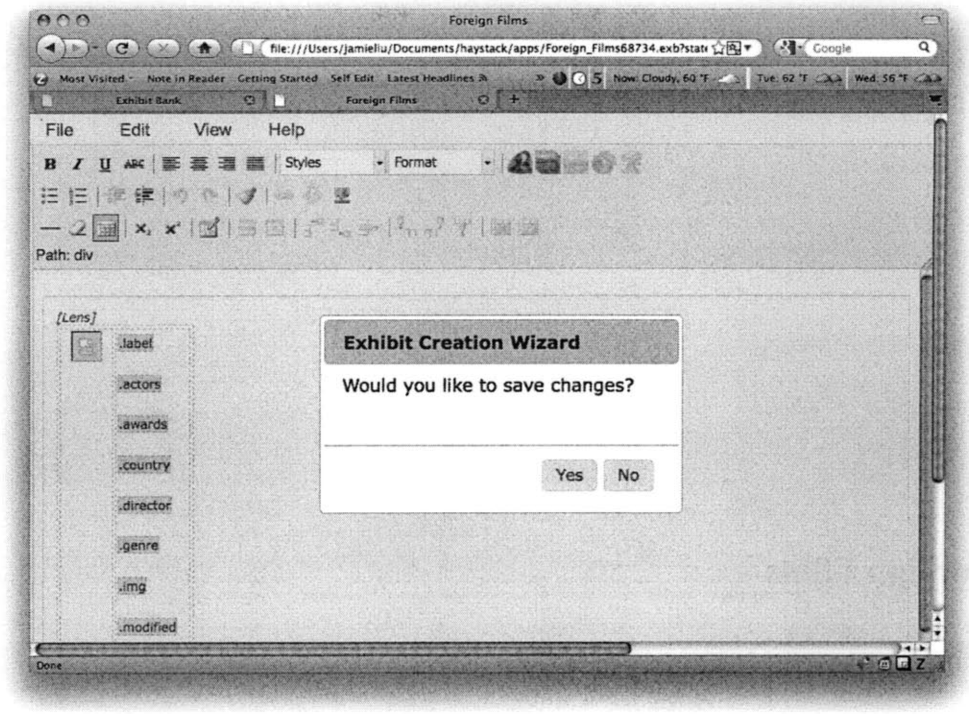


Exhibit Creation Wizard

Would you like to save changes?

Yes No

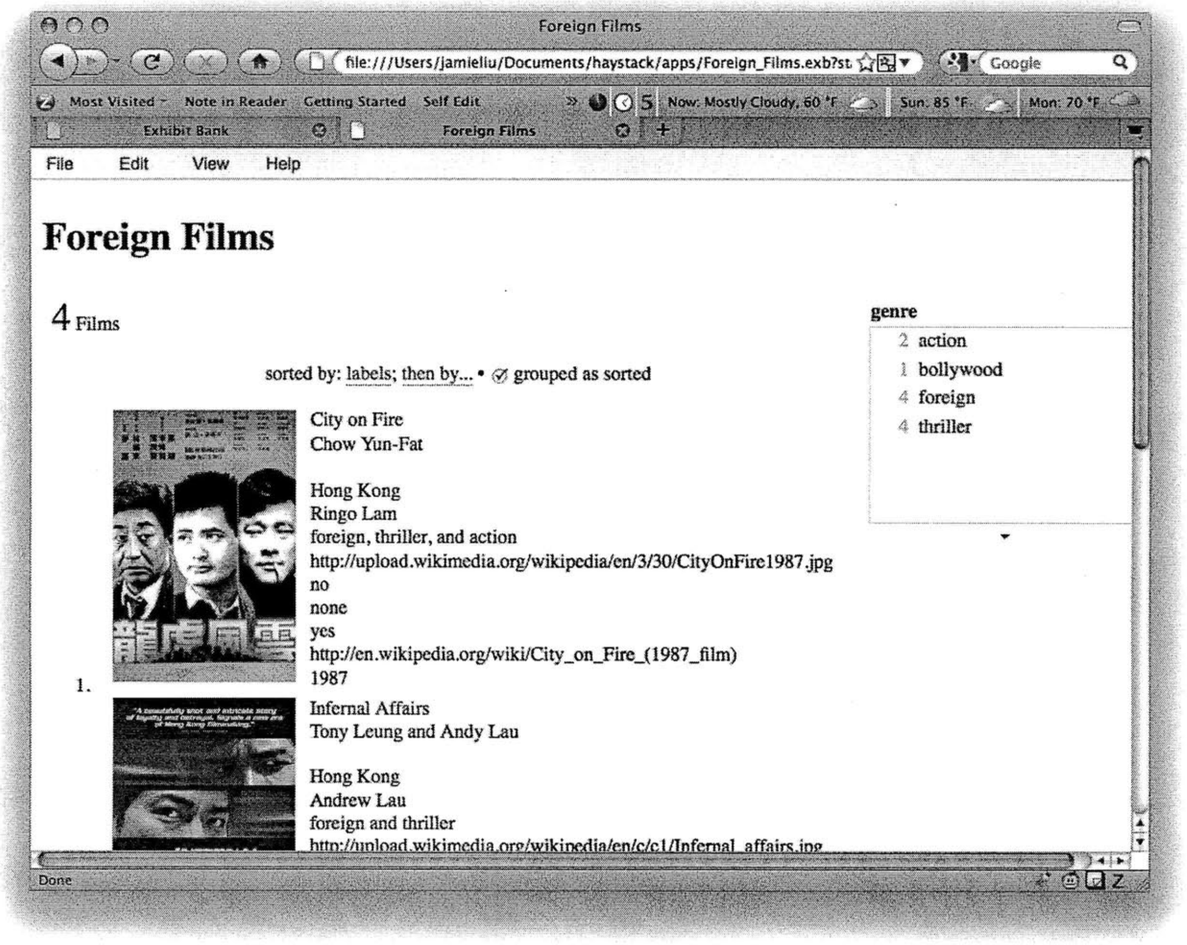
Figure 3-10: After the user selects a lens template, the lens is created and the user is prompted to save changes.

End of Getting Started

After Sally adds a lens, she is prompted to save all changes made during the course of the Getting Started wizard. Choosing to save writes the changes to disk and switches to the view pane to allow Sally to view the new exhibit (see Figure 3-11). Choosing not to save keeps her on the editor pane.

3.3 Editor Enhancements

The original Dido file placed all editing controls in one editor. This is a bit confusing to Sally for a few reasons. First, with the lens at the top of the editor content, the views and facets are obscured when initially opening the editor (see Figure 3-12). As a result, Sally may not get any immediate feedback on changes to either the view or editor when working with the Edit View and Edit Facet dialogs. While one solution may be to simply scroll down to the view and facet area when the user is making



Foreign Films

4 Films

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

1.

City on Fire
Chow Yun-Fat

Hong Kong
Ringo Lam
foreign, thriller, and action
http://upload.wikimedia.org/wikipedia/en/3/30/CityOnFire1987.jpg
no
none
yes
http://en.wikipedia.org/wiki/City_on_Fire_(1987_film)
1987

Infernal Affairs
Tony Leung and Andy Lau

Hong Kong
Andrew Lau
foreign and thriller
http://upload.wikimedia.org/wikipedia/en/c/c1/Infernal_affairs.jpg

genre

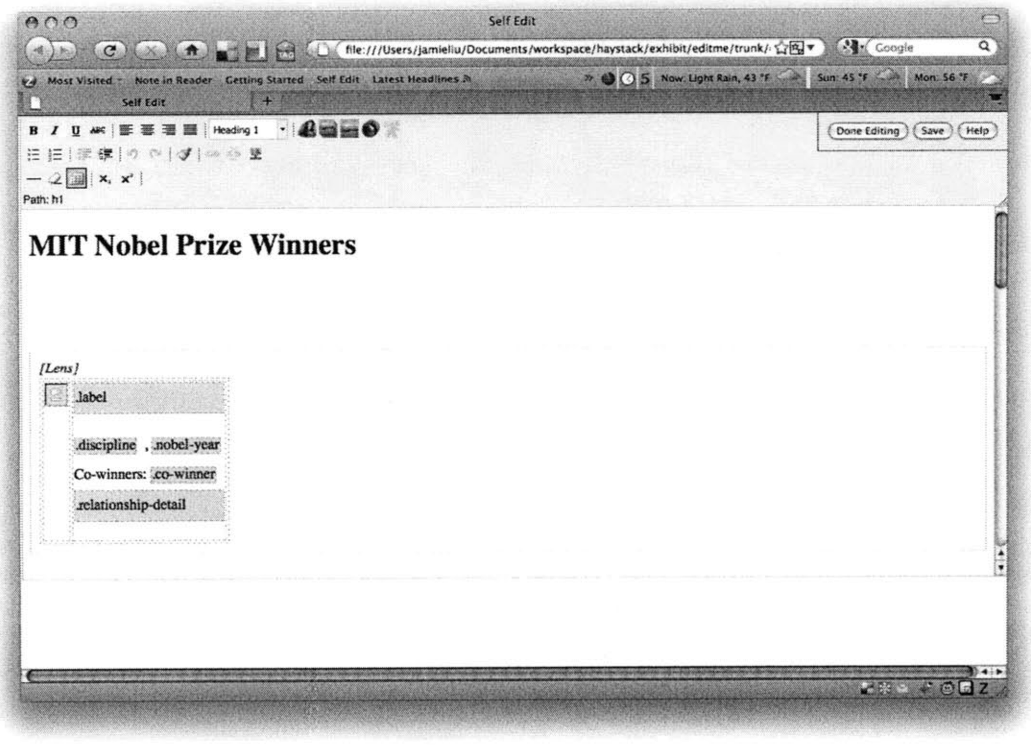
2	action
1	bollywood
4	foreign
4	thriller

Figure 3-11: If the user chooses to save changes, he or she is brought to the Exhibit Viewer to view the new exhibit.

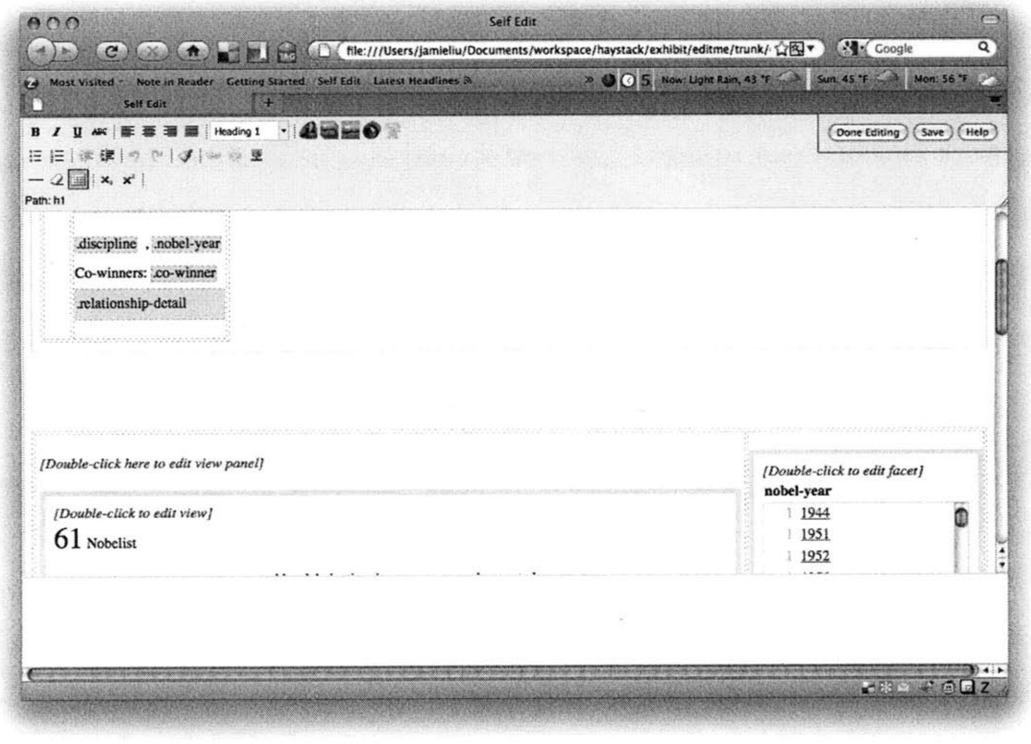
changes to those regions, a second and more important issue prompted the change to use two editors in two view panes instead.

There is a difference in mental model when working with lenses versus views and facets. Views and facets represent what the user will see in the resulting overall visualization. Adding a column of facets on the left and a view on the right results in a visualization that reflects that. On the other hand, lenses represent how each item will be rendered. Adding a single lens to the editor does not result in a single lens in the resulting display. Instead, the lens represents the visualization for each item that is repeated in the exhibit. With lenses appearing in the same editor as views and facets, it can be unclear that they have different effects on the resulting display. By separating them into two different editors, we clarify the purpose of the different UI components and Sally is less likely to be confused. In addition, the obscuring issue mentioned previously is resolved and the user gets instant feedback on any changes made without having to manually scroll past lenses to see the views and facets (see Figure 3-13).

A drawback to this design is that lenses are generic for all views. They cannot be scoped for specific views like they can in the original Exhibits described in Chapter 2. More work can be done to make this better, and this is described in Chapter 6 for future work.

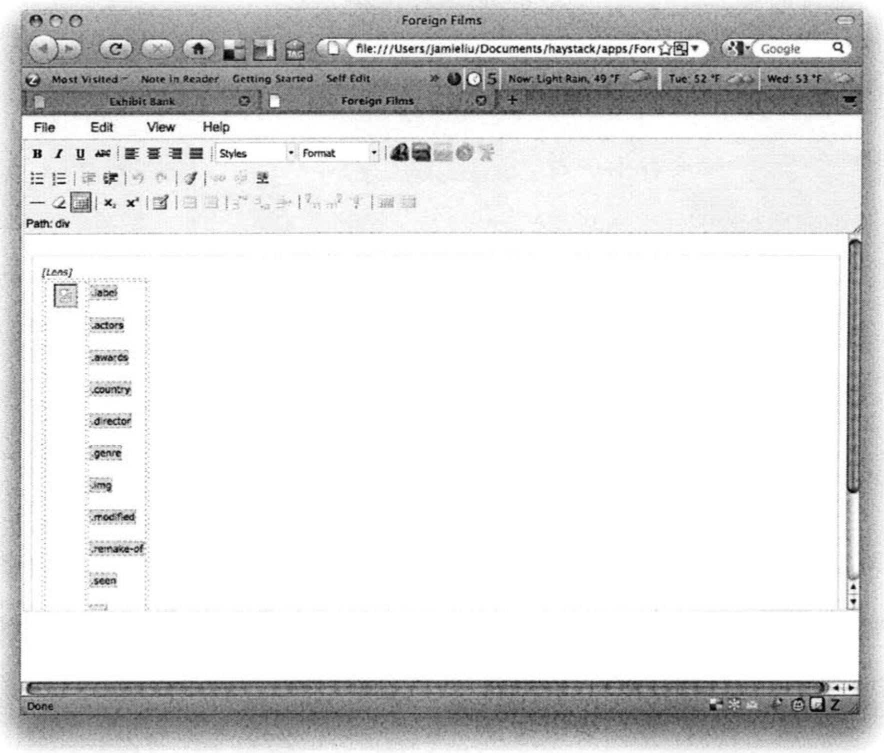


(a) When first opened, only lens is displayed

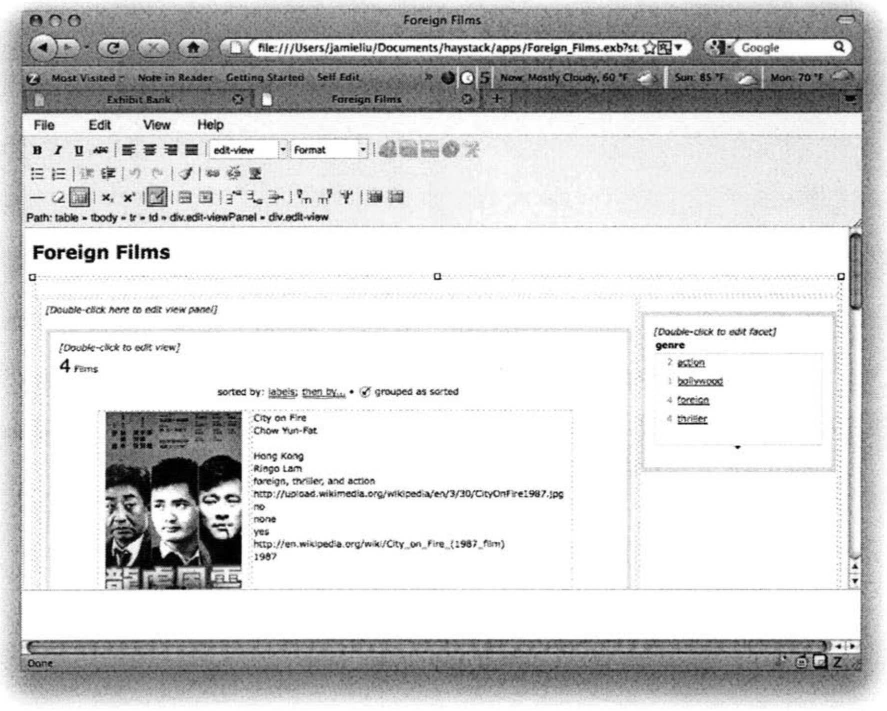

Self Edit
file:///Users/jamieliu/Documents/workspace/haystack/exhibit/editme/trunk/
Done Editing
Save
Help
Heading 1
Path: h1
.discipline , .nobel-year
Co-winners: .co-winner
.relationship-detail
[Double-click here to edit view panel]
[Double-click to edit view]
61 Nobelist
[Double-click to edit facet]
nobel-year
1944
1951
1952

(b) User must scroll down to see any views and facets

Figure 3-12: Original display editor



(a) New Lens editor


Foreign Films
file:///Users/jamieliu/Documents/haystack/apps/Foreign_Films.exb?st
Most Visited
Note in Reader
Getting Started
Self Edit
Exhibit Bank
Foreign Films
File
Edit
View
Help
edit-view
Format
Path: table » tbody » tr » td » div.edit-viewPanel » div.edit-view
Foreign Films
[Double-click here to edit view panel]
[Double-click to edit view]
4 Films
sorted by: labels; then by... • grouped as sorted
[Double-click to edit facet]
genre
2 action
1 bollywood
4 foreign
4 thriller
Done

City on Fire
Chow Yun-Fat

Hong Kong
Ringo Lam
foreign, thriller, and action
http://upload.wikimedia.org/wikipedia/en/3/30/CityOnFire1987.jpg
no
none
yes
http://en.wikipedia.org/wiki/City_on_Fire_(1987_film)
1987

(b) New View and Facet editor

Figure 3-13: New display editors

Chapter 4

Central Storage Support

In Section 2.2.2, we described the limitations of having a decentralized data store for each visualization a user may want to create with a given set of data. In this chapter, we describe the design and implementation of a central store, including the UI features necessary to support it. This central database gives users the opportunity to view the same data with multiple visualizations, without requiring maintenance of duplicate data. This eliminates the errors and inconsistencies that can arise when making changes to that duplicated information, and replicating those changes wherever the data is stored.

4.1 Data Store

There were several decisions made when considering how to store the data. In particular, we needed to decide where the storage system should be located – local versus on a server, what type of database to use, and the format in which to store the data.

There are benefits and drawbacks for having a data store on a user's machine versus having it hosted on a server. Over the years, an increasing number of server-based applications have gained popularity, with advantages of being able to access data from any machine with an internet connection and not having to install and maintain a desktop client application. However, local applications have the advantage of not requiring users to rely on an internet connection to access their information, and being generally faster at retrieving data because no calls need to be made to a server. In addition, users can safely store sensitive information locally without risking security flaws that may exist when storing data on a server. Dual storage options also exist that synchronize local data to a database on a server to allow users to access their data from both a desktop application and web application, and server-based applications have started to offer "offline" modes which allow users to save their data to their local machine for offline access. While offering both local and secure server-based storage is ideal, due to time constraints, only one storage option could be implemented during the time frame of this thesis project. Because the goals of this thesis involve helping users store any personal information, the advantages of accessing information securely and quickly with and without an internet connection outweighed the benefits of using a hosted database. Thus, the decision was made to first implement a local data store.

Because we wanted to leverage the features of Dido, we needed a way to save data accessed by a Dido HTML file. HTML5[6] plans to provide such capabilities with a local database that can be used to persist data, but it has not yet been officially specified nor implemented by modern web browsers. Thus, in the meantime, an alternative solution is needed. Of the leading open source database options available today, including SQLite, MySQL, and PosgreSQL, SQLite was the optimal choice for a local client application designed with a single user in mind because of its size and lack of configuration costs. In addition, Firefox already has installed a SQLite database wrapped in their Storage object[13], available to Firefox extensions. Google Gears[5], a browser extension compatible with most modern browsers, including Firefox, Internet Explorer and Safari[1], also offers SQLite database support in an add-on that the user can install. Because of the cross platform support that Google Gears offers and that Firefox's Storage lacks, Gears was the better solution for this project.

Using a browser's storage had the additional benefit of allowing users to get started using the application with as few barriers to entry as possible. Since all users who

[1]As of 2/19/10, the Google Gears team declared it would no longer be supporting Safari on OS X Snow Leopard and later (http://gearsblog.blogspot.com/2010/02/hello-html5.html).

have internet access already have a web browser installed, integrating the application with a browser's SQLite database would allow a user to avoid installing an additional desktop client application for the graphical user interface. Instead, users can use their existing web browser as the containing desktop client. Further, since users are already accustomed to browsing web-based applications, launching the system's interface in a browser would be a familiar experience.

Another important design decision was what data model to use. This thesis project gives users full control over what type of data is saved, as well as what specific properties of a data item are saved. For example, Graduate Student Sally can create a "Book" type to save off information about books read or books on her to-read list, and include any relevant properties, such as "title," "author," or "date read" vs "to-read." In addition to authoring the content, Sally can change the schema, or set of properties for an item, at any given time. As a result, there is never a pre-defined, concrete schema.

Supporting this dynamic data model can be done with either a standard relational database model or a Semantic Web[18] supported format, such as RDF[11]. The Semantic Web evangelizes a web where data is available free from presentation. This promotes the sharing and spreading of information, because all information is represented in a standard format. When using this data model, information is viewed as a network of objects and relationships. Each property can be saved as a "triple" containing subject, predicate, and object values, where the "subject" uniquely identifies the particular item, and the "predicate" and "object" respectively describe the relationship between the subject and the object, and the object value itself. Using this format, data can be combined from multiple sources without the distress of merging different schemas.

Alternatively, the standard relational database model uses tables for each type of item sharing a set of properties or schema. This model has the advantage of easily retrieving property values for a given item because all values exist in a single row of a table. When using an RDF model, the same query requires several `JOIN` statements. However, the RDF model has the advantage of more easily supporting the flexibility of not having a concrete schema while still saving structured information. With a triple store, a table does not need to be recreated each time the schema for a set of types changes. Because this application involves a dynamic set of data authored by the user, RDF was the chosen format.

Using a relational database, we can save the data in a three-column table with columns for subject, predicate, and object values. Each row is a "triple" representing one relationship between two objects. We create such a table in the Gears relational database to store all the data for the user's items. Because this type of data storage requires several relational database `JOIN` operations when aggregating all rows that pertain to a single subject, this type of data storage may not be practical when dealing with very large sets of data. However, for the purposes of this application, the data set was not expected to be large enough to impact the performance of the system. A "meta" table was also created in the database to store the information about each visualization created in the system. Such information includes the Dido file's URL, title, description, and date created.

4.2 Terminology

Before describing the system architecture of this application, we must first clarify the usage of the following terms:

- **Exhibit:** "Exhibit" with a capital "E" will be used to refer to the Exhibit project described in Section 2.1. With a lowercase "e," "exhibit" is used loosely to refer to any visualization produced by Exhibit. This includes the exhibits in standalone and anchored Dido files.

- **Dido or Dido file:** Dido and Dido file are used interchangeably to refer to the active document described in Section 2.2 that leverages Exhibit and offers a WYSIWYG editor to modify the visualization and data. These terms are used generally to describe the basic framework, and thus are used both when referring to standalone Dido files and an anchored Dido file.

- **Standalone Dido:** This term is used to refer to a traditional Dido file, which has its data stored in the file itself.
- **Anchored Dido:** This term is used to describe a Dido file that is not self-contained. Instead, the visualization is "anchored" to a central data store.

4.3 High Level Design

In this system, users can create any number of visualizations for data stored in a central repository. As described in Chapter 2, Dido's framework for allowing users to view and modify the data and display of an exhibit with a WYSIWYG editor is very applicable to this system. It gives the general end user the ability to change an exhibit without looking at any code. A goal of our system is to provide end users with such an interface for each visualization they create. However, before Dido can be leveraged, changes are necessary to make it work with the Google Gears backend.

Figure 4-1 shows the dependency relationship between the standalone Dido file and the anchored ones. The standalone, self-contained Dido file requires the modules depicted with sharp corners and tan background. These modules allow for the data to be stored in the file itself. The anchored Dido requires all modules depicted in the diagram. While primarily using the Gears storage, anchored Dido files still depend on the Self storage module to support export cabilities to standalone Dido files.

4.3.1 Storage Interface

Based on the architecture of Dido files, a generic Storage interface was created to associate different data stores with the rest of the Dido framework. This interface has two required functions, `getItems()` and `saveItems()`. For the case of a standalone Dido file, the SelfStorage object is used to retrieve items from and save items to the file itself. For an anchored Dido, a GearsStorage object is used to retrieve items from the Gears database based on a query associated with the file and save items back to the database. This pluggable Storage model was designed to be extensible, such that

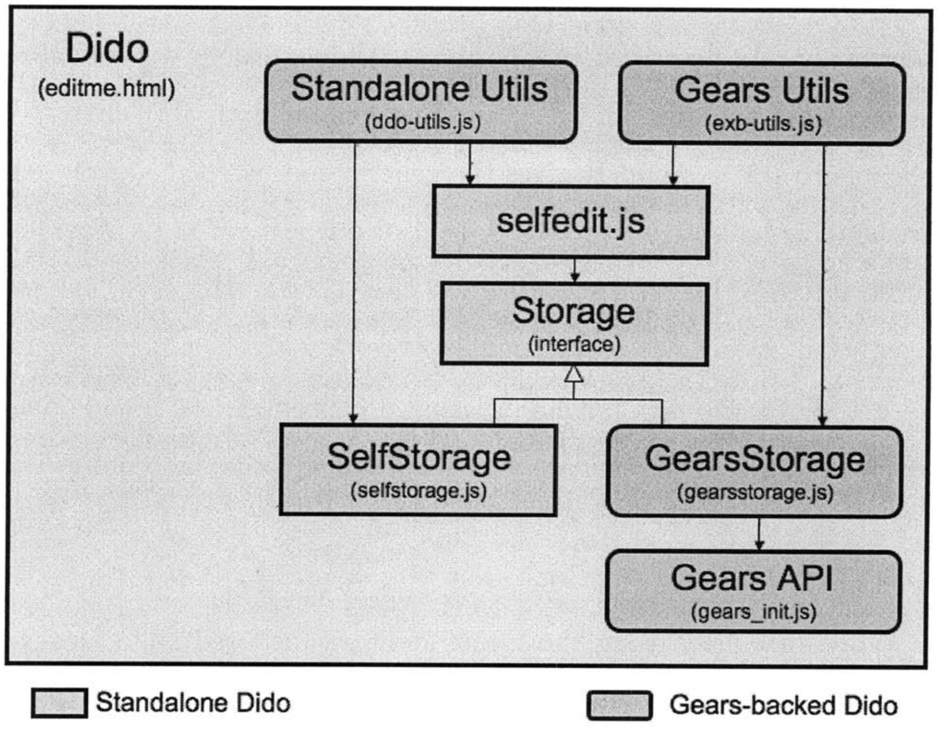


Figure 4-1: Dependencies for Standalone Dido and Anchored Dido

if other Storage objects are created in the future, they can easily be plugged into the Dido framework.

4.3.2 Utility Modules

In order for standalone Dido files and anchored Dido files to coexist, a plug-in approach is needed to ensure that the necessary functionality is added to the Gears version without interfering with the standalone version. As a result, two utility objects are created to support initializing an anchored Dido file, as well as an exported standalone file. These utility modules ensure that the file is properly initialized with the correct Storage object, and injects necessary functionality specific to the respective file types. For example, for an anchored Dido file, the Gears utility inserts a user interface to allow users to modify the query associated with the file. Such a UI would not make sense in the case of the standalone Dido, and thus gets "plugged in" only for the anchored Dido case.

4.4 User Interface Features

This chapter describes the design decisions behind all the user interface components added to this system to support the centralized data store. The primary features include giving users the ability to add data to the information repository and create visualizations to view that data. In addition, as mentioned in Section 4.3, in order to leverage Dido, work still needed to be done to integrate it with the Gears backend. As part of that work, users needed to be able to select what data to include in each visualization and be able to modify that selection.

Based on the goals of the system, there are three different areas that required development. These areas are described in detail in the subsequent sections of this chapter.

1. **Dido Manager:** The user interface that allows users to manage existing exhibits.

2. **Dido Creator:** The user interface that gives users the ability to create a new visualization by specifying what data should be included.

3. **Plug-ins to Dido:** Features necessary to add to Dido to support the Gears backend.

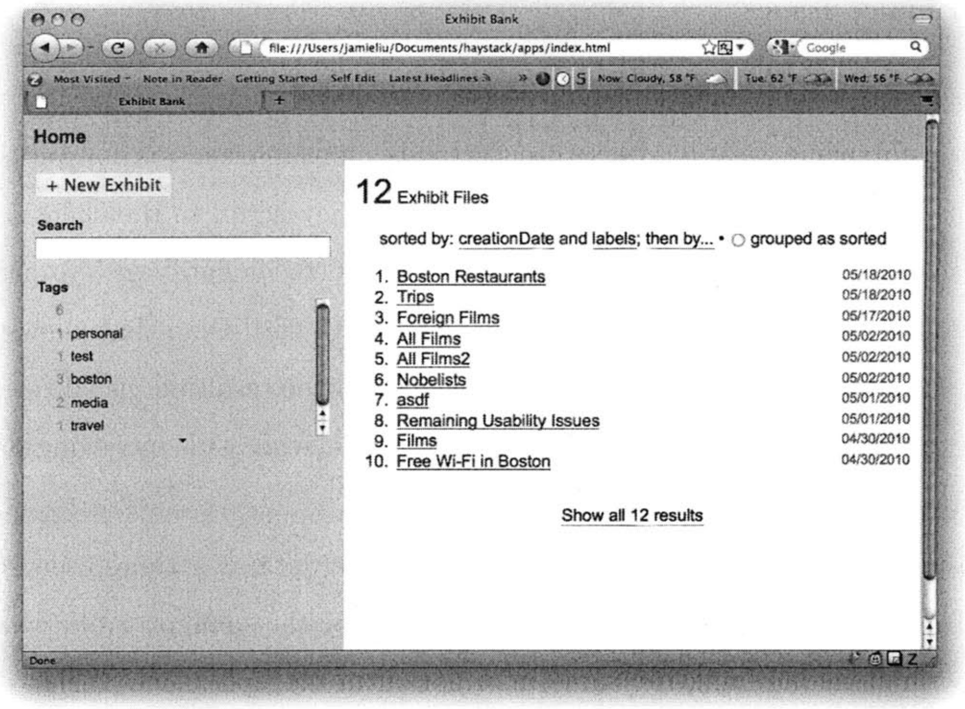
4.4.1 Dido Manager

Users of this system can create any number of visualizations. For each visualization they create, a new Dido file is created to provide them with the appropriate management capabilities. Graduate Student Sally, our persona from Chapter 1, needs to be able to manage all the visualizations she creates. We introduce a home page to allow her to view in one place all the visualizations she has already created. This home page is simply another web page with an Exhibit that allows Sally to interact with her Dido files.

When first getting started, Sally will either want to open an existing visualization or create a new one. Because creating a visualization is a one time process, Sally is more likely going to be working with an existing one. Thus, we have created a home page (Figure 4-2) that lists the existing exhibits with links to launch them.

There are a few features of the home page worth noting. First, a prominent "New Exhibit" button exists in the top left to allow Sally to immediately click and create a new exhibit if that is her desired action. Next, because this application encourages Sally to create any number of visualizations for any number of data types, it has the potential to house many exhibits. When this is the case, Sally needs a mechanism to quickly find a particular visualization. A visually familiar navigator is added to the left of the list to help her filter the exhibits based on tags, and an incremental search text field is added above the navigator to allow her to search by any text in the title or description of the exhibit. The exhibits are initially sorted by the descending order of creation date, so Sally can see the most recent documents first.

Each exhibit title links to the Dido file. When designing the system, it was considered whether to launch the visualization in a new tab in the browser or to



6	
1	personal
1	test
3	boston
2	media
1	travel

12 Exhibit Files

sorted by: creationDate and labels; then by... • ○ grouped as sorted

1.	Boston Restaurants	05/18/2010
2.	Trips	05/18/2010
3.	Foreign Films	05/17/2010
4.	All Films	05/02/2010
5.	All Films2	05/02/2010
6.	Nobelists	05/02/2010
7.	asdf	05/01/2010
8.	Remaining Usability Issues	05/01/2010
9.	Films	04/30/2010
10.	Free Wi-Fi in Boston	04/30/2010

Show all 12 results

Figure 4-2: Home Page

launch the visualization in a new tab within the application. We chose the former approach for a few reasons. First, it reduces the amount of management needed to be done within our application itself. Second, the browser's management of files provides features that could be useful for an end user, including opening tabs in new windows and bookmarking URLs to return to a particular visualization later. Further, launching documents in new tabs is a familiar concept for anyone who has used web applications such as the popular application suite for Google Docs[2].

4.4.2 Dido Data Selector

When creating a new Dido, Sally needs to choose what items to include from the data repository. In this section, we explain the necessity for adding a query creator to represent the items Sally wants in her visualization, as well as the query creator's user interface. We also describe the new features for adding and importing new data to the system.

[2]http://docs.google.com

Performance Issues

Because there is only one data repository to store all items, the repository has the potential for being very large as the user adds more information for various types of items. Because Exhibit manages all the data in javascript in the client's browser, it becomes slow when thousands of items are added to its database. To handle this scaling issue, we introduce a "query" associated with each Dido that allows users to specify a subset of the repository data to include in the resulting visualization. This allows the visualization to continue to perform well, while still retrieving data from a central store.

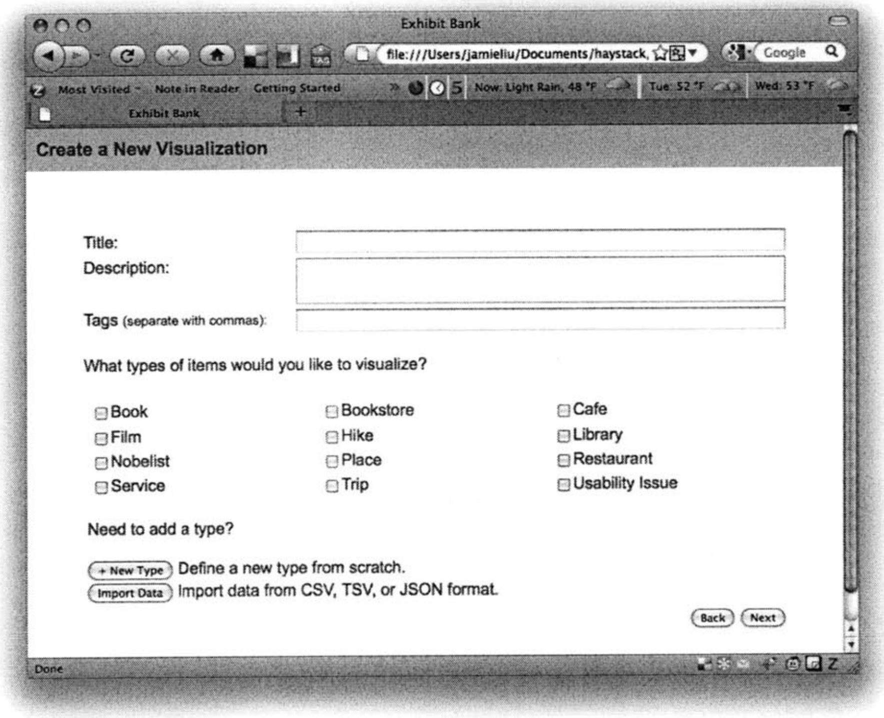
Despite only being able to view a subset of the data at a time, users still have the advantage of creating multiple visualizations for the same or overlapping set of data without maintaining the same information in multiple places. This decreases inconsistencies and errors that can arise when maintaining changes to the same data duplicated in different places. With a central store, Sally can change a colleague's name in her "All Contacts" Dido file, and see the modification reflected in her visualization for "All researchers." Moreover, users generally manage different information types with different visualizations, therefore specifying such a query does not severely detract from the usability of the system. For example, Sally typically will want to separate her management of media items (DVDs, Books, etc) from her list of contacts.

To support the query, we add an interface for users to specify what items to include in the visualization when they create a new Dido, as well as an interface in the resulting Dido file that allows users to modify that query after the visualization has been created.

Query Creator

A basic and common use case for specifying which items to include in a visualization is the set of all items of a certain type. For example, Sally may want to create a visualization of all Person items or of all Book items. This is generally how people organize their information, therefore we optimize the user interface for this case, but provide additional functionality for specifying a more complex set of data.

As shown in Figure 4-3, a list of checkboxes is populated for the various types of items in the database. Users can select one or more types. In addition, once the user has selected a type, it appears in a sidebar on the left. The type has a “refine” link. Clicking on that link will expand the type and show all properties for that type given the items in the database. Each of the properties can be expanded to reveal the possible values. Sally can refine the data set by selecting any of those values (see Figures 4-4 and 4-5). This gives users the power to be as specific as they'd like in deciding what data they want to include in their visualization. For example, Sally can create an exhibit of all films, or an exhibit specific to Foreign dramatic films.



Create a New Visualization

Title:
Description:
Tags (separate with commas):

What types of items would you like to visualize?

☐ Book	☐ Bookstore	☐ Cafe
☐ Film	☐ Hike	☐ Library
☐ Nobelist	☐ Place	☐ Restaurant
☐ Service	☐ Trip	☐ Usability Issue

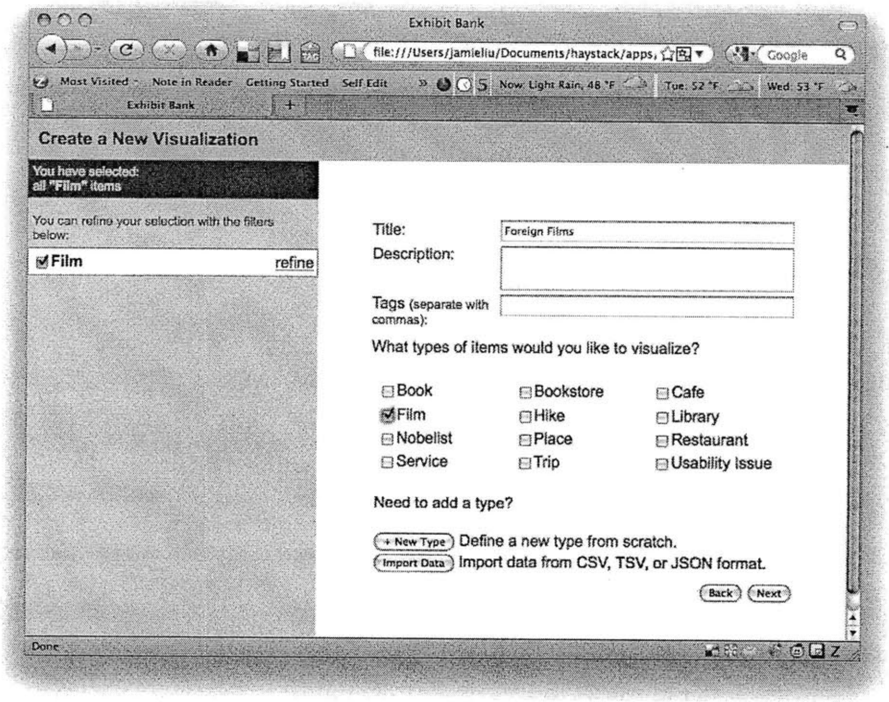
Need to add a type?

+ New Type Define a new type from scratch.
Import Data Import data from CSV, TSV, or JSON format.

Back Next

Figure 4-3: Dido Creator

When Sally clicks "Next", a query object is saved to the file, representing the selection of objects that she has chosen. This query object saves off which types are selected, and for each type, any facet values that are selected. When the anchored Dido opens, it loads items from the gears storage, using the query created based on this information.



Create a New Visualization

You have selected:
all "Film" items

You can refine your selection with the filters below:

☑ Film refine

Title: Foreign Films

Description:

Tags (separate with commas):

What types of items would you like to visualize?

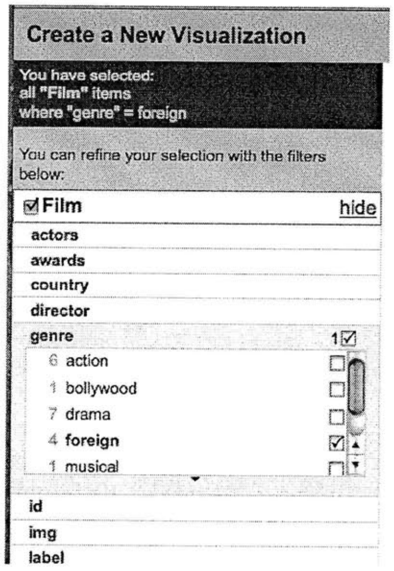
☐ Book	☐ Bookstore	☐ Cafe
☑ Film	☐ Hike	☐ Library
☐ Nobelist	☐ Place	☐ Restaurant
☐ Service	☐ Trip	☐ Usability Issue

Need to add a type?

+ New Type Define a new type from scratch.

Import Data Import data from CSV, TSV, or JSON format.

Figure 4-4: Dido Creator - Select type



Create a New Visualization

You have selected:
all "Film" items
where "genre" = foreign

You can refine your selection with the filters below:

☑ Film hide

actors
awards
country
director
genre 1☑

6	action	☐
1	bollywood	☐
7	drama	☐
4	foreign	☑
1	musical	☐

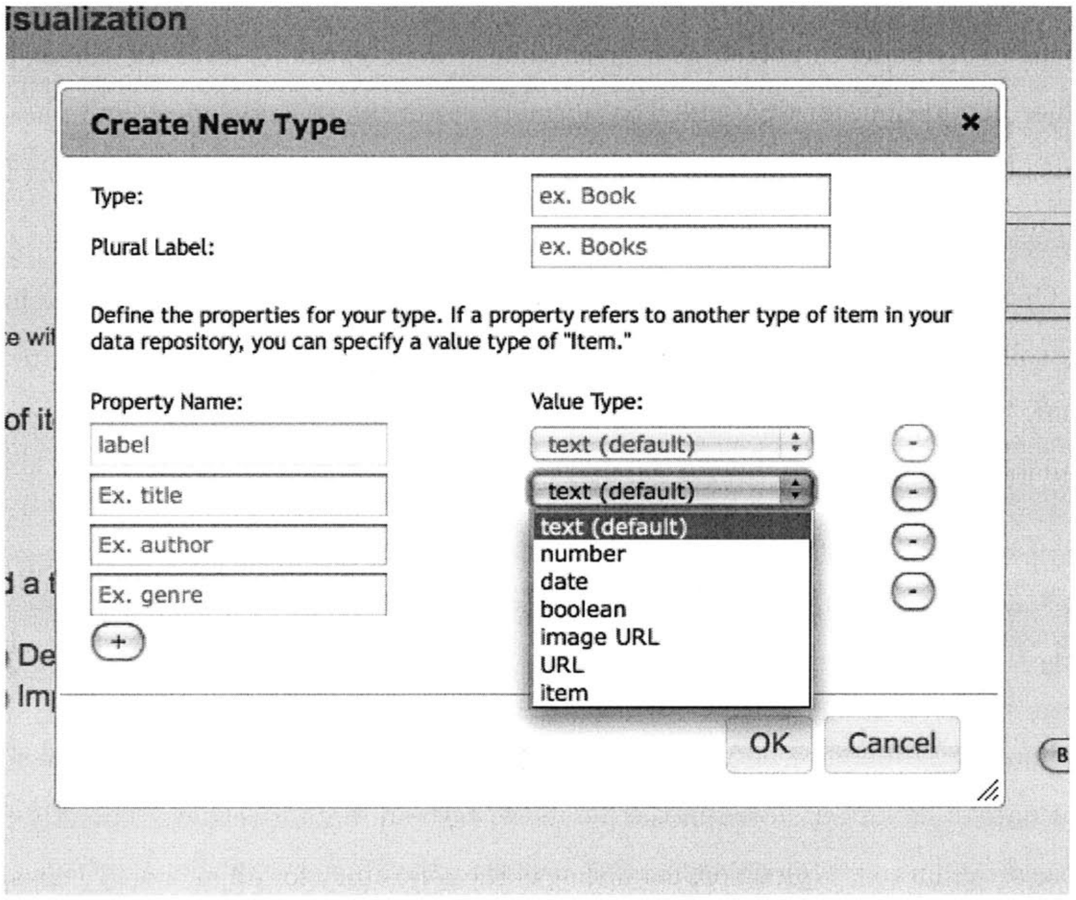
id
img
label

Figure 4-5: Dido Creator - Refine type

Adding Data

Sally, as a new user, needs a way to add data to the system in order to visualize that data. There are two possible scenarios in which she would need to add data: the data is already stored somewhere or she is adding new data from scratch. In the former case, Sally will want a mechanism to import the existing data into the system. In the latter case, she will need to be able to specify what type of object she is adding and what properties it has. We provide solutions for both.

When defining a new type, users can specify the type label, plural label, and any properties they want to associate with the type (see Figure 4-6). This information is added to the Exhibit types repository for users to then add items for that type once they are working with the anchored Dido file.



Create New Type

Type: ex. Book

Plural Label: ex. Books

Define the properties for your type. If a property refers to another type of item in your data repository, you can specify a value type of "Item."

Property Name:	Value Type:
label	text (default)
Ex. title	text (default)
Ex. author	
Ex. genre	

Figure 4-6: New Type dialog

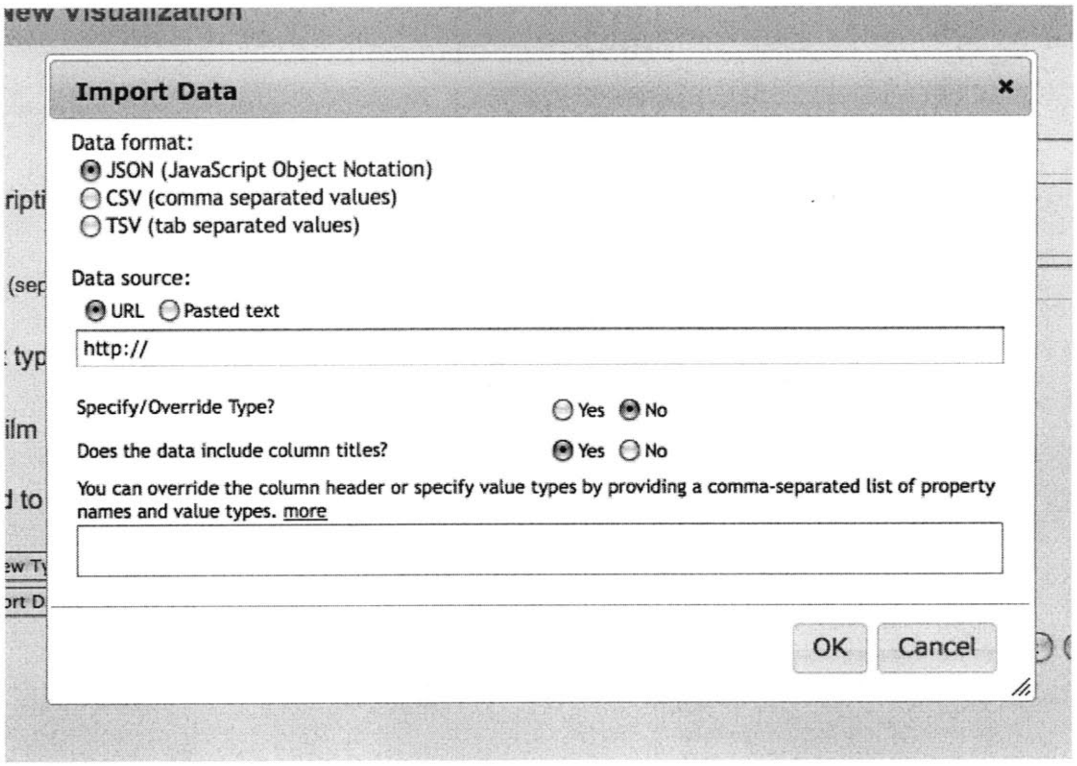
Users can also import data from another source. In designing this application, it was important that users had a means of importing information that they may have stored elsewhere. Users may want to jump-start or supplement their collections of information by scouring existing data sets available on the web or in local files. Existing Exhibits are associated with a JSON data file that users could simply point to. However, because the typical end user may not have data already in JSON format, importing other formats was a necessary feature. Since spreadsheets are a common tool that people use to store information, it was important to provide import mechanisms for the two data formats frequently used when exporting or copying data from a spreadsheet: CSV (comma separated values) and TSV (tab separated values).

Currently, the system supports copying any JSON, CSV, or TSV formatted text to be imported to the system, as well as importing data from remote JSON or TSV files. Users can specify a type label and plural type label (ie. "Book" and "Books") for all items being imported at a given time if none exists, or override the existing value. In addition, for CSV and TSV formats, users can override any column headers and specify property value types, such as text, url, or number. A sample import form can be seen in Figure 4-7.

Once Sålly has a visualization already in the system, she can add new items to the collection directly from the data editing interface. That is, if she wants to add a new book to her Book collection, she can simply use the "Add New Item" option available in Dido.

4.4.3 Plug-ins to Dido

The primary difference between a standalone Dido file and an anchored Dido file is that data is no longer stored in the file itself. Instead, data is retrieved directly from the central database (Google Gears) that is the repository for all anchored Didos. As a result, there are several places where we plug into the existing Dido framework to provide functionality now available or necessary for this model. These changes are described in this section.



Data format:
[✓ JSON (JavaScript Object Notation)]
CSV (comma separated values)
TSV (tab separated values)

Data source:
[✓ URL] Pasted text
http://

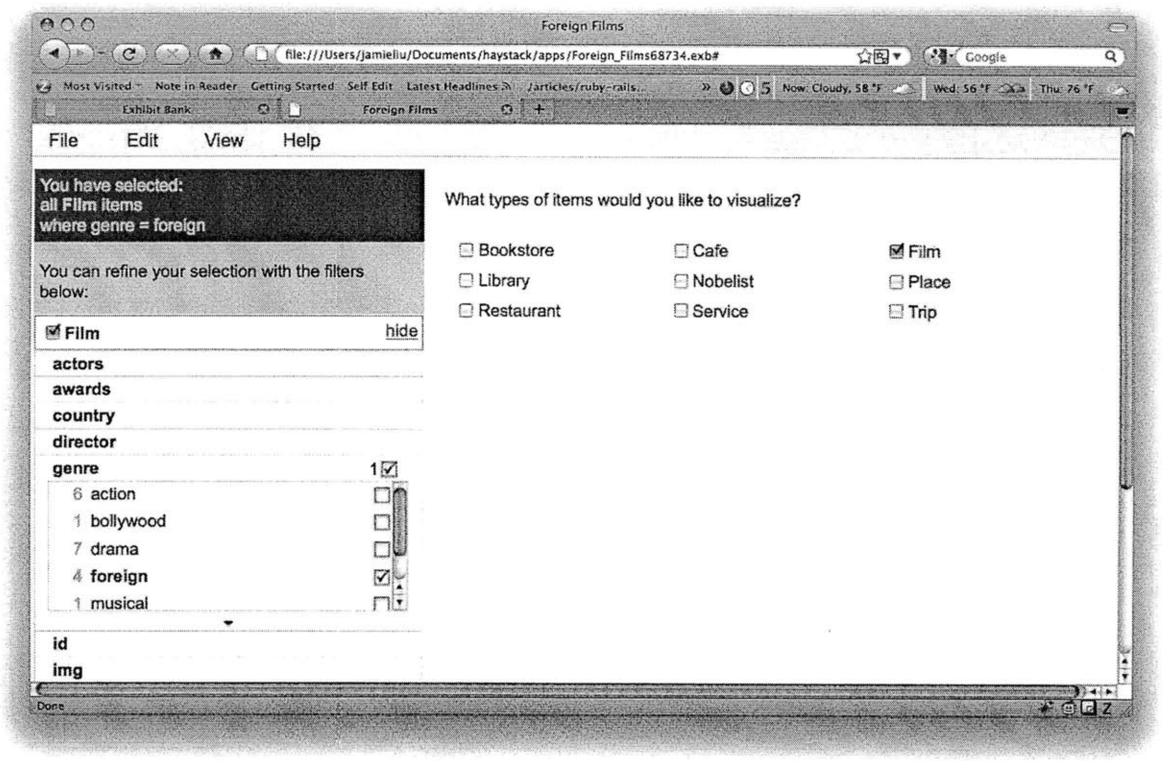
Specify/Override Type? Yes [✓ No]
Does the data include column titles? [✓ Yes] No
You can override the column header or specify value types by providing a comma-separated list of property names and value types. more

Figure 4-7: Import dialog

Query Support

As mentioned in Section 4.4.2, each anchored Dido is associated with a query that describes what items from the database should be added to the anchored Dido. This query is generated after Sally selects which items to add to her visualization when creating a new exhibit. Sally may forget to refine her data selection or choose to do so after creating a visualization. To support this, we provide another viewer in the resulting Dido file for modifying the query. This query editor, known to end users as the "Data Picker," is shown in Figure 4-8.

The UI controls are loaded when the user selects the "Data Picker" from the View menu dropdown, and is refreshed each time such that it always reflects the state of the exhibit. The controls displayed are laid out in the same manner as those on the initial anchored Dido creator page to allow users to immediately understand what they need to do to make changes.



You have selected:
all Film items
where genre = foreign

You can refine your selection with the filters below:

[✓ Film]

genre 1 ✓

- 6 action ☐
- 1 bollywood ☐
- 7 drama ☐
- 4 foreign ✓
- 1 musical ☐

What types of items would you like to visualize?

☐ Bookstore	☐ Cafe	✓ Film
☐ Library	☐ Nobelist	☐ Place
☐ Restaurant	☐ Service	☐ Trip

Figure 4-8: Data Picker

Exporting Data

Once Sally has created a visualization for her collection of items, she may need to share it or display it for others to see. In some cases, she will want to share her data along with presentation, but in other cases, Sally may want to share only the raw data. Thus, the application must support both exporting an anchored Dido to both a standalone Dido file as well as raw JSON, CSV, or TSV data.

An Export menu (Figure 4-9) has been added to the global File menu to provide this functionality. All anchored Dido files can be exported as a standalone Dido file in which a new Dido is created with the data stored in the file itself. As a result, the file is self-contained and can be emailed or hosted on a server for others to view. The menu also provides the actions to export the raw data as JSON, CSV, or TSV data for users to copy and save to a file or paste elsewhere.

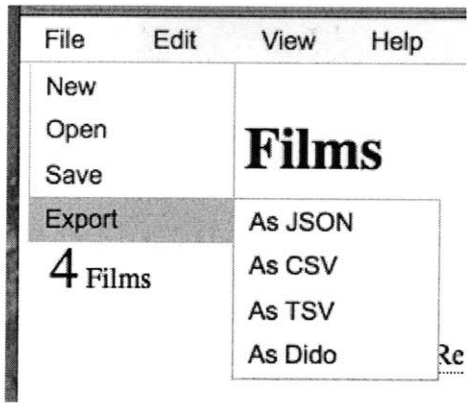


Figure 4-9: Export options

Links Back

Finally, the following actions have been added to the File menu to take users back to the other UI areas:

- **File - Open** launches the Home page in a new tab.
- **File - New** opens the New Exhibit form in a new tab.

4.5 Summary

In this chapter, we discussed the various design decisions made for both the back-end and front-end of the system in order to support the Dido framework with a centralized storage system. A "query" specifying what data should be included with each visualization was required for performance reasons, and thus the necessary UI for creating and modifying that query were crucial components added to the system. Finally, a plug-in architecture is needed because the user interface changes described in Section 4.4.3 do not make sense in a standalone Dido file. As a result, we dynamically inject the additional UI components to the standalone Dido at runtime without modifying the core code.

Chapter 5

Evaluation

The application described in this thesis is designed for the layman end user, who may or may not have any programming experience. Users of this system should be able to create visualizations with their own data, customize how to view and browse their information, and share their information with others without writing a single line of code or HTML. We evaluate this system through usability and user study tests. The *usability* test informs us of how effective the system's user interface is in terms of the features it is designed to support, while the *user study* evaluation indicates the system's potential viability as a solution for the problems addressed in this thesis. In this chapter, we discuss the system's successes and failures for each respective assessment.

5.1 Usability Results

Five sit-down usability testing sessions were conducted, followed by a survey and interview. Users were asked to perform the following tasks:

1. From the home page, launch an exhibit.
2. Create a new exhibit from data already in the system. Filter the data (ie. Select Academy Award winning films rather than all Films).
3. Customize the visualization by specifying what type of view to use and what

facets to add. This is open-ended; there are no restrictions or guidelines about what the end result should be.

4. If the user did not add a facet, ask them to add facets to filter by genre and country.
5. Modify the label for an item.
6. Add a new item to the exhibit.
7. Create a new exhibit from imported data.
8. Export the exhibit as a standalone Dido file and send it to a friend.

The testing confirmed that we had taken the right approach in helping users manage their different visualizations and get started using the Dido editors, but also exposed some areas where *getting the correct design* proved to be a difficult task. There were several challenges in designing the user interface for this application, primarily because the system is a hybrid between a document and application editor. Since it does not fit a single mold, creating an interface familiar and natural to users was not trivial. In the end, the greatest strengths of this system were in areas where our application was able to maintain UI consistency with other applications, as well as places where the user did not have to "think" to perform the desired task. In this section, we walk through the key user interface components that make up this system, and describe what did and did not work, as well as what was learned about the design space during the process.

5.1.1 Dido Enhancements

In this section, we discuss the usability of the enhancements made to all Dido files, including the menu bar, getting started wizard, dialogs, and editors.

Menus

Overall, the switch to using a menu bar over a control panel was successful, but the location of some menu items can be reevaluated. Users were familiar with the menu interface, which had common global menu items, File, Edit, View, and Help.

Standard menu items, such as Open, New, Save, and Export were found immediately, while others, such as the various Edit-related items, sometimes took a few tries. When performing a task, users almost always first looked for actions in the menu bar before looking in the toolbars. In fact, multiple users commented that it would be useful to have the custom toolbar actions (for modifying the visualization) duplicated in the menu bar for better discovery.

The location of the menu items for showing the editor panes appeared to be natural for most users, but some expected those menu items to be under a "Tools" or "Edit" global menu because those are more likely to have actions that correspond to the task the user is trying to accomplish. In other words, when the user wants to edit the visualization, he expects to find a tool to help him do so, rather than look for a different "view" to display. Having the editor panes under the "View" menu reflected the model behind the system. In order to edit the visualization, the user must switch to view the editor panes. However, if a user formed a different mental model for how the system works, he or she may not think to "go" to a different "view" in order to perform those actions.

The "Edit Data" and "New Item" actions were found immediately, but three users tried to edit the data in the View and Facet editor first. They had mistaken the preview of the view, which displayed the actual data, to be an area that they could edit. Further, when in the editor panes, the data editing menu items are disabled, and it took some users a few tries to realize they needed to be in "view mode" for those actions to be enabled. This is another case where the mental model of the users is different from the system model. If users thought that all editing could be done in the editor pane, they would not think to switch views to edit the data. Adding better feedback, such as enabling the action when in an editor view and showing text explaining that the user needs to switch views in order to edit the data, can help users understand the system better. Alternatively, we can enable the action and automatically switch views for the user when he or she tries to access it.

Getting Started wizard

The Getting Started wizard was successful at guiding users in the process of creating a complete visualization of the data. While producing an exhibit based on a particular design was not tested in this exercise, we observed that the wizard allowed users to add each necessary component for the resulting visualization, starting with choosing a view, followed by adding any number of facets, and concluding with picking a lens template.

In initial user testing, it was clear that users did not know what some configuration options meant, primarily when creating a view. Many of them went to the reference page or used the default options and hoped that they could proceed smoothly. Some minor enhancements were made to ameliorate this issue, such as adding a "view picker" to help users quickly see what the available types of views were, as well as providing better UI controls for entering values. Hover help for all configuration options would further improve a user's understanding of the different settings, and would make the technical (non-user-friendly) reference page obsolete.

The lens template picker allowed users to specify an initial layout for the relevant properties associated with the data types involved in the exhibit. Although the default option was the existing or blank lens template, all users chose an option that actually generated a pre-populated lens. This indicates that the lens template picker was a useful addition, because users did not want to start with the blank slate. Most users chose the "Landscape" option, which allowed them to specify an image property. Once selected, most users hesitated when looking at what to enter when given the property dropdown list for selecting an image. This may be due to the fact that for this task, we were creating a visualization for data already in the system, and thus users were unfamiliar with the properties listed in the dropdown and did not know which value they should pick. This could also be due to lack of text explaining what type of value they should select. In any case, this was certainly an area where the user had to hesitate before continuing, and thus can be improved in the future.

Overall, the wizard created a flow of dialogs such that all users were able to successfully create a visualization. With improved UI controls for setting different configuration values, as well as ensuring that proper default values were set, users were able to proceed through the wizard successfully.

Editors

We had separated the editors into two different editor panes, one for modifying the lens and one for modifying the overall visualization, including views and facets. While we did not test the initial editor, which contained both the Lens and View and Facet editors, this modification seemed to be a good decision because users knew which editor to go to when making changes to the overall visualization (ie. they knew to use the View and Facet editor when adding a facet). The original editor would have made the user spend more time thinking about where to insert different UI components, and from general observations during the user testing, it was clear that users wanted to do the least amount of "thinking" during the process of creating the visualization.

While the wizard helped users add views, facets, and lenses initially, once outside the wizard, users had trouble finding the equivalent actions. Some users took an extensive amount of time to find the toolbar buttons that allowed them to add the various UI components, and needed assistance along the way. Two users ignored the toolbars entirely because at quick glance, they appeared to be standard toolbars and they did not realize custom toolbar items existed. Conversely, one user immediately noticed the custom toolbar items *because* the toolbars were standard and the custom actions stood out. However, this last user is most likely the exception because he was already familiar with tinyMCE[17] editors. The discoverability of these actions may be improved by lowering the toolbar items on the toolbar to the last row, where users first look for custom actions. In addition, multiple users suggested duplicating the actions in the menus. Finally, most users were able to figure out that they must first place focus in the display before clicking on an action to add a UI component, but some required assistance with this, as well. One way to respond to this is to allow the UI edit actions to be available at all times, such that if the user tries to add a UI component before placing focus anywhere, he or she is warned and given instructions on how to proceed.

We did not explicitly test the lens editor – the main interaction with it was the dynamic creation of a lens based on the user's selection during the wizard – however, based on users who did spend some time playing around with it, it appeared that they had trouble understanding how to use it. This is due to the fact that the lens editor model was unique. The content behaved like text at times, but had some "variables" that are replaced in the rendering of the visualization. Users did not immediately suspect that they could edit existing lens content (ie. the variables) by double clicking on it. Some users appeared to discover this functionality accidentally. In addition, users did not realize that by selecting content in the editor, they could drag it around. One user expected that clicking on the yellow lens content and dragging would allow them to rearrange the content, rather than selecting the content first. While typically, user feedback gestures can help guide a user in understanding how items can be manipulated, the standard practices are already applied in this case. The cursor is an I-beam to indicate that the contents can be treated like text, and there are no "gripper" or "texture" marks to indicate that the lens content can be "clicked and dragged." Because standard feedback is already used, better help text may be the best approach at describing what users can and cannot do in the editor. In addition, a preview pane that shows a sample rendered item may give users the proper feedback to understand how changes to the lens affect the actual rendering of an item.

Overall, users were able to modify the overall visualization once they figured out they needed to place focus in the editor and where to find the toolbar items, but most required guidance at some point. As mentioned earlier, duplicating some actions in the menus and additional help text would improve the discoverability of available actions. This area should be explored further in future design iterations and testing.

5.1.2 Central Storage Support

In this section, we will review the usability of the features added to specifically support anchored Dido files. This includes the Dido manager, creator, and query editor (aka.

"Data Picker").

Dido Manager

The Dido Manager was straightforward for all users. They were able to launch exhibits from the home page and bringing up the New Exhibit form. This is likely because the home page was similar to typical list view interfaces, such as that for an inbox or the home page of Google Docs[4].

Dido Creator

The Dido creator interface, which allows users to specify basic Exhibit properties, including the title and any tags, as well as which data types to add to the visualization, appeared to be mostly straightforward, as well, because all users were able to successfully create an exhibit for "Film" items. Some users did not realize "types" were required, in which case they were shown error text explaining that they must select a type in order to continue. Perhaps separating these input options into two steps, one for filling out basic information and one for selecting data types, would help clarify the necessity for selecting types.

An early implementation of this interface gave users the ability to first choose which properties they wanted to use to filter the data, followed by an interface to select values for those properties. For example, users could select "type" and "city" as properties on which to refine the data, and later specify particular values (ie. type values of "Restaurant" and "Place" types, and a city value of "Boston"). Being given all properties for all items of different types was evaluated as too confusing for layman end users, and thus the interface was changed to allow users to first consider what type of items to add, followed by any filtering. This was the right decision, because users understood the concept of having different "types" of items without feeling overwhelmed by too much data.

Some users did not notice the "refine items" area that appeared in a left sidebar once the user selected a type. This area provided a summary of what items were selected, and gave users the opportunity to further filter the selection based on facet values for the type. In our testing, we asked users to create an exhibit for "Academy Award-winning films." The left sidebar provided an area for the user to refine the selection appropriately. All users who noticed the area were able to refine the data properly. Other users ignored the section entirely. The sidebar was designed to appear once the user made a selection in order to draw attention to the sidebar. It was also designed to handle the typical case of creating an exhibit for all items of a certain type, by hiding the facet options by default. It is possible that the collapsing of the facet values contributed to users ignoring the area all together, thinking that it did not offer any additional functionality aside from deselecting a type that had been selected. Better UI feedback can help with this, perhaps by showing a "+" sign to indicate the section can be expanded, rather than the "Refine" link.

While the import dialog was clear for some, for others it was a bit unclear what was expected in the properties. As a result, users often went with the default values. For example, users are able to import CSV or TSV data, and are given the option to overwrite the "type" and column headings for the items. All users left this blank because they did not know what it meant. This was an issue because it was never pointed out that the term, "type", as well as "label," had special meaning in Exhibits. As a result, it did not occur to users that there would be a problem if their data did not have a "label" column header or that all items would be categorized as generic "Item" types if no "type" column header was specified. This was immediately understood after an explanation, therefore additional error checking and more precise, instructive help text will likely help resolve this issue. In addition, a more descriptive import "wizard," as opposed to a single dialog, would help guide the user in specifying the data properly. The wizard could parse the data and prompt the user to specify a "type" if none existed. It could also ask the user to specify a column that represents an item's "label" if such a column did not exist. Further, if the user is importing data types that already exist in the system, the import wizard can display the existing properties associated with that type and help the user align mismatched schemas.

Anchored Dido Features

In the actual Dido file, there were a few UI components added specifically for the anchored Dido. These include the "Data Picker," which allowed users to modify their initial selection of types, as well as menu items to bring the user back to the Dido manager home page and the Dido creator page.

Those who used the Dido data picker were able to change their selection successfully. The users who did not initially filter the data when creating the exhibit were able to do so properly with the Data Picker. Thus, the addition of this UI was important for anyone who did not initially specify the correct data set. Some users confused the Data Picker with the facet editor, when asked to add a facet to the visualization that filters Films based on "genre." This confusion was somewhat understandable, given that both UI components involve filtering the data. All users who made this mistake eventually realized they were in the wrong place to accomplish the task at hand, but some took longer than others to realize this. More help text explaining what the Data Picker UI does, along with a reference to the View and Facet editor, might help prevent this issue from happening in the future.

Exporting data was clear to users, again due to the consistency with external applications. Similarly, the "Open" and "New" links back to the Dido Manager and Dido Creator were self explanatory for the same reasons. Users are simply accustomed to seeing these menu items under the File menu, and thus immediately knew where to look for these actions.

5.1.3 Summary

The system was able to successfully guide users in creating a visualization for data. The strengths of the system were in places where users did not need to "think" to get the job done. This was often in areas that were consistent with external applications, as well as areas that provided enough UI guidance. Standard menu items were found immediately, and the Getting Started wizard gave users clear enough choices or proper default values to allow them to quickly create visualizations.

Weaknesses of the system were primarily in areas where our application differed from standard applications, and there existed a disconnect in the system's model and a user's mental model of the system. In fact, the user who was able to complete all tasks successfully and quickly attributed it to being able to form a correct mental model about how the system worked. This user was also already familiar with faceted browsing and tinyMCE, which may have influenced his thought process. Our application differs from typical ones, primarily because it contains many "editable" properties – users can edit the data values themselves, the visualization that displays the data, as well as *what* data types are shown. As a result, providing an interface that is learnable and usable is a challenging design space. When continuing to design UI components, we must put more emphasis on what a user is thinking when they want to accomplish a task over how the system is designed.

The structure of the usability testing conducted here focused primarily on the learnability of the system. Overall, the issues found here seem fixable given more time and resources to perform more design iterations, implementation, and usability testing. In addition, a more involved user study should be done in the future, based on actual usage of the system over a period of time. More details on how the system is used in a "real world" setting, including which features are most commonly accessed and any usability issues that may be revealed after frequent use of the system, are some factors that will govern how this application evolves and improves in the future.

5.2 User Study Results

After users gained experience using the system during the usability session, we interviewed them to collect information on possible applications of the system, including what information they might store and any limitations that would prevent them from using the tool. Overall, it was clear that there was an interest and necessity for such an application. Users gave several personal use cases for why this application would be an effective tool. However, despite the desire for such a system, there were also features that stand in the way of users adopting it.

5.2.1 Use cases

All users stated that they could imagine using such a system. While this question may have been "leading," evaluating their response time for providing a sample use case showed that most users were able to think of examples immediately (hesitation of less than 1 second). The most noteworthy potential use cases were the following:

1. **Startup Companies** A graduate student wanted to track startup companies in which she was interested. She currently used a spreadsheet to manage the information and found the faceted browsing and visualization options of our application to be a useful feature to help her navigate the data more effectively. With faceted browsing, she can quickly view the companies by location or product.

2. **Classical Music collection** A post-doc interviewed wanted to use this application for her classical music collection. She found iTunes to be limited in the metadata fields it provided for each file. She wanted to specify more precise fields for her CDs, such as specifying a "Conductor" for orchestral pieces, rather than forcing that information into the "Artist" field for iTunes.

3. **Sports Statistics** A young professional thought the faceted browsing would be quite useful for maintaining sports statistics. In particular, by storing all the context information for a given play, the faceted browsing would help in viewing the most frequent causes for bad plays.

4. **Biomolecular Sequences** A graduate student in bioengineering wanted to organize the DNA sequences he commonly works with, and found existing tools to be restrictive. Currently using LabLife[1] to organize his data, he appreciated the customizability of our system for giving him control over what values to store that LabLife did not provide. However, since LabLife offered more tools out of the box, it would take more for him to migrate to using this application. This is discussed further in Section 5.2.2.

[1]http://www.lablife.org

5.2.2 Barriers

When exploring what limitations the application currently has, it was clear that there were some features that users wanted before they would transition to using the system.

1. **Automatic Import of Metadata** Despite supporting the importing of data from popular formats used in spreadsheets and Exhibits, a popular request was to have a better automatic import system. Files that already have metadata associated with them, such as documents, photos, and music files, should have that data automatically added to the system. For example, in the classical music example, the user wanted to be able to insert a CD and import all information about the album and music files to the system. In addition, importing data to this system from others needed to be better supported. Currently we provide migration from spreadsheets and Exhibit JSON files, but users can be organizing their data with other tools. As one user phrased it, "having a good import feature is important to get users to convert to one system from another, because people who tend to use these [organizational] tools often already have their own system."

2. **Analytical Tools** Adding support for calculations based on property values was another request. Currently, the system allows users to sort, search, and filter on any given property, but it does not allow for the same rich interactions based on analyses of the data. For example, while users can specify location information, there is no facet to allow them to search for all items within a certain distance of a given address. Another example is with DNA sequencing. Some properties can be determined based on analyses of the data, such as the melting temperature. Such information is provided by third party applications that specialize on managing this information. Thus, while the user sees the additional flexibility of this customizable system, without the ability to do similar analyses, he would not want to migrate. There may already exist some analytics support in Exhibit, and this should be investigated and better exposed in the

future.

3. **Community** Another major demand was to have a system on a server that could support contributions from multiple people, along with the appropriate access control moderations. In this case, the user wanted to manage data he himself could not be the sole content author, but wanted the shared data to be somewhat moderated. More work can be done to further explore this space, including looking into the possibility of leveraging existing tools such as Google Docs or Semantic Media Wiki. With a shared Google Doc, users can create their own Dido file for visualizing the data, while leveraging the compiled data.

5.2.3 Summary

The interviews conducted were designed to gauge initial thoughts on the utility of such a system. This assessment confirmed that this application helps solve a real need for end users, who want more effective ways of managing and browsing their data. However, it also indicated that some limitations do exist which may prevent users from migrating to the system. These barriers to entry should be looked at in the future to help encourage adoption of this application.

Chapter 6

Conclusion

In this thesis, we've introduced a new tool that enables end users to more effectively organize their collections of information by creating personalized and interactive interfaces for managing their data. We describe the motivation behind creating such an application, the design and implementation details of our system, and an evaluation of our tool's usability, potential viability, and current limitations. A summary of the contributions made in this thesis are presented in this chapter, as well as a few directions in which this project can be pursued further with future work.

6.1 Contributions

The system developed here offers an improved end-user driven customizable information management tool. We introduced a central data store on the user's local machine for saving the user's data, an interface for creating and managing visualizations for that data, and a wizard and editors for customizing the visualization and editing the data. The system provides a means of importing data to help users migrate existing data from other applications, as well as gives users the opportunity to define new data objects. Our application also encourages the sharing of data either as raw data or with an accompanying visualization.

As a first iteration for such a tool, more enhancements can be made to improve the tool's usability, as discussed in Chapter 5. However, in its current state, this tool is successful at conveying the ideas behind the system and allowing users to create more effective interfaces for curating their data.

6.2 Future Work

There are several areas related to this project that can be explored further. Some are described in detail in Section 5.2.2, and will not be repeated here. Additional areas that are especially noteworthy are discussed below.

6.2.1 Database

There are two specific database enhancements that can be looked at in the future. The first is a replacement of the current local database storage implementation as a result of the evolution of web standards, while the second is the addition of a server side database.

HTML5

During the course of work on this thesis, the Google Gears team declared that it would no longer support future enhancements of Gears in an effort to focus on the adoption of the HTML5 standard[6]. HTML5 has specifications to support a local database, such that the functionality offered by Gears will become obsolete. Once that specification is stable and web browsers fully support the local database, implementation of the database wrapper for this system should be moved to using the HTML5 local database API instead of that for Gears.

The Cloud

While using a local database has its benefits, it has its disadvantages, as well. First, the information can only be accessed from a user's local machine. By adding a server side database, users could have a back up of their data and be able to access their data and visualizations from any computer that has internet access. As mentioned in

Chapter 4, an ideal system would synchronize the data on the user's local machine with the one in the cloud such that when a user does not have internet access, he or she can work with locally stored data, and when the user does have access, any changes made locally while offline are synchronized with the data on the server. Adding support for cloud storage may go hand-in-hand with the changes needed to support HTML5.

In addition, as mentioned in Chapter 5, support for a server-based storage facility that can manage different users and access controls may be required for certain use cases. Thus, when implementing server support, this should be taken into consideration.

6.2.2 Advanced Lens support

The lens of a resulting visualization is a critical part of this application, as it is a template rendered repeatedly for each item. Better support can be added to make the customization of lenses easier and more powerful for end users.

Scoped Lenses

In the traditional exhibits described in Chapter 2, lenses can be scoped for specific views. In other words, an exhibit that has multiple views can use a different lens for each view. This functionality was removed in Dido, which limits the expressive power of the visualizations.

Data Editing support

With the current system, when users edit or add new data, they are only given text fields to work with. This can limit a user's efficiency of using the system, and is more prone to typos, as well. Offering other types of form fields, such as drop down lists with existing values for a property or radio and checkboxes for selecting options, would be more effective for the end user. This is also an interface more familiar to users for data entry.

Manipulation and styling of lens content

As discovered in user testing, manipulating the arrangement of lens content and styling the overall lens was not the most straightforward task. Work can be done to investigate tinyMCE capabilities more to improve these actions.

6.2.3 Anchored Dido and Standalone Dido

Finally, in this thesis we talk extensively about the support for two types of Dido files: Anchored (with a central storage) and Standalone. There is currently limited support for transitioning between the two. While users can import and export data to migrate data between the two, there is no concept of automatic synchronization. Should all files store information in both the file itself and Gears? Should Standalone files automatically import data to Gears if the user has it installed? This can be an area to explore further to create a better ecosystem in which the two types of files can coexist.

Bibliography

[1] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[2] Douglas Crockford. The application/json media type for javascript object notation (json), July 2006. `http://www.ietf.org/rfc/rfc4627.txt`.

[3] Filemaker, Inc. `http://www.filemaker.com`.

[4] Google Docs. `http://docs.google.com`.

[5] Google Gears. `http://gears.google.com`.

[6] Ian Hickson. HTML5: A vocabulary and associated APIs for HTML and XHTML. `http://dev.w3.org/html5/spec/Overview.html`, April 2010.

[7] David F. Huynh, David R. Karger, and Robert C. Miller. Exhibit: lightweight structured data publishing. In *Proceedings of the 16th international conference on World Wide Web*, pages 737–746, Banff, Alberta, Canada, 2007. ACM.

[8] iTunes. `http://www.apple.com/itunes/overview/`.

[9] David R. Karger, Scott Ostler, and Ryan Lee. The web page as a WYSIWYG end-user customizable database-backed information management application. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 257–260, Victoria, BC, Canada, 2009. ACM.

[10] H. Kilov. From semantic to object-oriented data modeling. In *Systems Integration '90. Proceedings of the First International Conference on Systems Integration*, pages 385–393, Morristown, NJ, USA, 1990.

[11] Frank Manola and Eric Miller. Rdf primer. In *http://www.w3.org/TR/rdfprimer/*, 2003.

[12] Microsoft Excel. `http://office.microsoft.com/excel`.

[13] Mozilla Storage. `https://developer.mozilla.org/en/Storage`.

[14] Scott B. Ostler. Mutable data representations for lightweight publishing and curation. Master's thesis, MIT, 2007.

[15] Picasa. `http://www.google.com/picasa`.

[16] Tim Berners-Lee on the next Web, 2009. `http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html`.

[17] TinyMCE: Javascript WYSIWYG Editor. `http://tinymce.moxiecode.com/`.

[18] W3C semantic web. `http://www.w3.org/2001/sw/`.