

VISUALIZING THE UNIVERSAL DATA CUBE

A Dissertation Presented

by

CURRAN KELLEHER

Submitted to the Graduate School of the
University of Massachusetts Lowell in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2014

Computer Science

© Copyright by Curran Kelleher 2014

All Rights Reserved

VISUALIZING THE UNIVERSAL DATA CUBE

A Dissertation Presented

by

CURRAN KELLEHER

Approved as to style and content by:

Dr. Haim Levkowitz, Chair

Dr. Jesse Heines, Member

Dr. Rosane Minghim, Member

Dr. Jie Wang, Department Chair
Computer Science

ACKNOWLEDGMENTS

I would like to thank my parents Daniel and Marcella Kelleher and my brother Sean for supporting me throughout my education. I am grateful to my committee members Dr. Haim Levkowitz, Dr. Rosane Minghim, and Dr. Jesse Heines for their support and feedback throughout the dissertation process. I would also like to acknowledge Dr. Georges Grinstein and Dr. Daniel Keim for their mentorship and support of my work in its early stages.

ABSTRACT

VISUALIZING THE UNIVERSAL DATA CUBE

SEPTEMBER 2014

CURRAN KELLEHER

B.Sc., UNIVERSITY OF MASSACHUSETTS LOWELL

M.Sc., UNIVERSITY OF MASSACHUSETTS LOWELL

Ph.D., UNIVERSITY OF MASSACHUSETTS LOWELL

Directed by: Professor Dr. Haim Levkowitz

The field of data visualization is lacking open tools that support easily developing and using production quality interactive visualizations. Particularly, there is a need for reusable solutions for (1) well known visualization and interaction techniques (2) authoring and sharing visualizations with multiple linked views, and (3) describing existing data such that many data sets can be easily integrated and visualized. This dissertation introduces novel data structures and algorithms for interactive visualizations of data from many sources, addressing these three issues.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	iv
ABSTRACT	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
 CHAPTER	
1. INTRODUCTION	1
1.1 Vision	3
1.2 Challenges	9
1.3 Application Areas	11
2. REACTIVE VISUALIZATIONS	13
2.1 Related Work	13
2.2 Reactive Models	17
2.3 Reactive Bar Chart	24
2.4 Reusable Reactive Flows	26
2.5 Linked Views	31
2.6 Summary	34
2.7 Future Work	34
3. COLLABORATIVE VISUAL DATA EXPLORATION	37
3.1 Related Work	37
3.2 Application State Model	39
3.3 Runtime Engine	41
3.4 Changing State	42
3.5 Real-time Synchronization	44
3.6 History Model	46
3.7 Integration with Web Technologies	46

4. THE UNIVERSAL DATA CUBE	47
4.1 Related Work	48
4.2 Case Study: Causes of Death.....	52
4.3 Data Set Representation	58
4.4 Querying Data Sets	63
4.5 Integrating Data Sets	64
4.6 Limitations of Data Cube Representation	66
4.7 Proof of Concept	69
4.8 Crowdsourcing Data Experiment	72
4.9 Summary	75
5. CONCLUSION	76
5.1 Contributions	76
5.2 Future Work	78
5.3 Visualizations and Interactions	80
5.4 Data Sources.....	81
5.5 A Grammar of Graphics Approach.....	87
5.6 Final Thoughts.....	88
 APPENDICES	
A. PSEUDOCODE CONVENTIONS	90
B. OPEN SOURCE PROJECTS	95
 BIBLIOGRAPHY	 98

LIST OF TABLES

Table	Page
2.1 Reusable flows for reactive visualization.	28
4.1 A Cube Data Set about Population.	59
4.2 A Cube Data Set about GDP.	59
4.3 A Concordance Data Set linking equivalent terms used by different cubes referring to countries.	59
4.4 Examples for UDC Concepts.	61
5.1 Reusable Data Cube Visualizations.	82
5.2 Reusable Data Cube Visualizations (continued).	83

LIST OF FIGURES

Figure	Page
1.1 Millennium Development Goals User Interface.	4
1.2 CIA World Factbook User Interface.	5
1.3 GapMinder Data Access User Interface.....	6
1.4 CDC Vital Statistics User Interface.....	7
1.5 NSF Bachelors Degrees Statistics Choropleth Map.	8
1.6 Fragmentation of Data and Visualization.	10
1.7 Bridging the Gulf between Data and Visualizations.	10
1.8 Introducing Intermediate Representations.	11
2.1 Full Name Reactive Model Example.....	24
2.2 Reactive Bar Chart Flow	27
2.3 Reactive Line Chart	30
2.4 Configurable Scatter Plot	30
2.5 Linked Scatter Plot and Bar Chart.....	32
2.6 Linked Views Flow	32
2.7 Improved Linked Bar Chart, Scatter Plot and Table	33
2.8 Reactive Table Visualization	36
3.1 GapMinder and Hans Rosling.....	38
3.2 Nested Box Layout Configuration	40

3.3	Nested Boxes with Visualizations	40
3.4	The Rapid7 Ingress Dashboard.	42
3.5	Dashboard Scaffold Configuration Editor.	43
4.1	Universal Data Cube Overview	48
4.2	CDC Mortality Visualization Version 2.....	54
4.3	Cardiovascular disease raw tree.	55
4.4	Cause of Death Tree Visualization.....	56
4.5	CDC Mortality Visualization with Linked Views.	57
4.6	UDC Logical Data Structure	62
4.7	UDC Data Integration Algorithm	66
4.8	United Nations Population Prospects Data Set	69
4.9	World Population Timeline	70
4.10	World Bank GDP Data Set	71
4.11	Scatter Plot of Integrated Data	72
4.12	Linked Line Chart and Choropleth Map	73
4.13	Mechanical Turk Results	74
5.1	New York Times Ebola Visualization	86
B.1	Model-Contrib Home Page	96

CHAPTER 1

INTRODUCTION

The field of data visualization is lacking open tools that support easily developing and using production quality interactive visualizations. While powerful visualization and data analysis technologies exist such as Tableau [68], Spotfire [78], SAS Visual Analytics [79] and Wolfram Alpha [101], they cannot be freely leveraged as tools by researchers or components for application developers due to their proprietary nature. The primary open source tool for Web-based data visualization is D3.js [19], a JavaScript library that provides low level primitives for constructing interactive visualizations. Creating visualizations using D3 typically involves copying one of the many visualization examples available, then tailoring it to work with the application-specific tasks and data at hand. Though there have been many attempts at creating generalized visualization environments using D3, none of them rival the power of commercial visualization packages.

Open Source projects often serve as a bridge between academic theory and professional practice. For example, Hadoop has made parallel computation more accessible, and PostgreSQL has served as a platform to bring advanced database techniques into practice. D3 has served as a bridge between data visualization theory and practice largely via the self-contained examples found in the D3 Example Gallery [21]. However, most D3 examples are not reusable solutions but rather one-off implementations tailored to a specific data set and task. There is a need for an approach to generalizing D3 examples that results in reusable interactive visualization components. Chapter 1, “Reactive Visualizations” introduces a solution for generalizing existing visualization

examples using concepts from functional reactive programming and the Model View Controller paradigm.

Some of the most common requirements of visualization applications today are that they have multiple linked views, can be developed collaboratively, and can be easily shared. Having linked views means that interactions in one visualization cause some change in other visualizations on the page. Being developed collaboratively means that many people in a group are all able to make changes that can subsequently be leveraged by others in future work. Being easily shared means that the resulting visualizations can be embedded within presentations, reports, or third party Web sites. Chapter 2 “Collaborative Visual Data Exploration” discusses an approach to collaboration, history and linked views that leverages reactive visualizations.

Much of the work of a data scientist or analyst is dedicated to curating and transforming data such that they can perform their analyses. Many data sets, in aggregated form, can be transformed into data cube using binned aggregation [100]. Data cubes are data structures based on dimensions and measures. Dimensions contain (potentially hierarchical) sets of distinct entities or concepts. Measures are quantitative properties that represent aggregates (such as sum or average). Many data cube dimensions, such as Space, Time, Gender and Industry, are “universal” in that they transcend any single data set. Observations (sometimes called facts) within a data cube assign concrete numeric values to measures of the data cube corresponding to the cartesian product of members from each dimension. These dimensions may be referenced by different data sets using different identifiers, and data sets may present the same measure with a different scale or attribute naming convention. Chapter 3 “The Universal Data Cube” introduces a collection of data structures and algorithms for integrating and visualizing many data sets.

Taken together, the contributions of this dissertation make inroads toward an open platform for complex data visualization. The hope is that these technologies

can continue to evolve with support from the global Open Source community and can evolve into a mature platform for advanced data visualization applications.

1.1 Vision

The envisioned data representation and visualization framework can serve as a digital telescope into the universe of phenomena on Earth via publicly available data. For example, consider data sources such as the United Nations, the US Census, the US Bureau of Labor Statistics, or the US Centers for Disease Control. These organizations and hundreds of others around the world provide publicly available data about various topics including population statistics, public health, distribution of wealth, quality of life, economics, the environment, and many others. By unifying these data sources and providing users with tools to explore them visually, a deeper understanding of the world can be gleaned by anyone through the lens of data.

There is immense potential value in data that is not being realized. The ability to visually explore data lends itself to applications in education, journalism, and public policy. Especially in the era of “Big Data,” it is increasingly valuable for organizations and individuals to have the ability to analyze large quantities of data that come from various sources and vary across time, space, and other dimensions. In addition, publicly available data can provide context for business-centric, proprietary data analysis activities.

While publicly available data sets are available on the Web, it is difficult to realize their full value in practice. The difficulty stems from the fact that they are made available using numerous formats and protocols. The heterogeneity of formats and protocols used makes it difficult to combine and analyze data sets together and hinders the development of analysis and visualization tools. For example, some data sets are made available as CSV files, Excel spreadsheets (as shown in figure 4.8), or must be navigated using a Web-based user interface (as shown in figures 1.1, 1.2, 1.3 and 1.4).

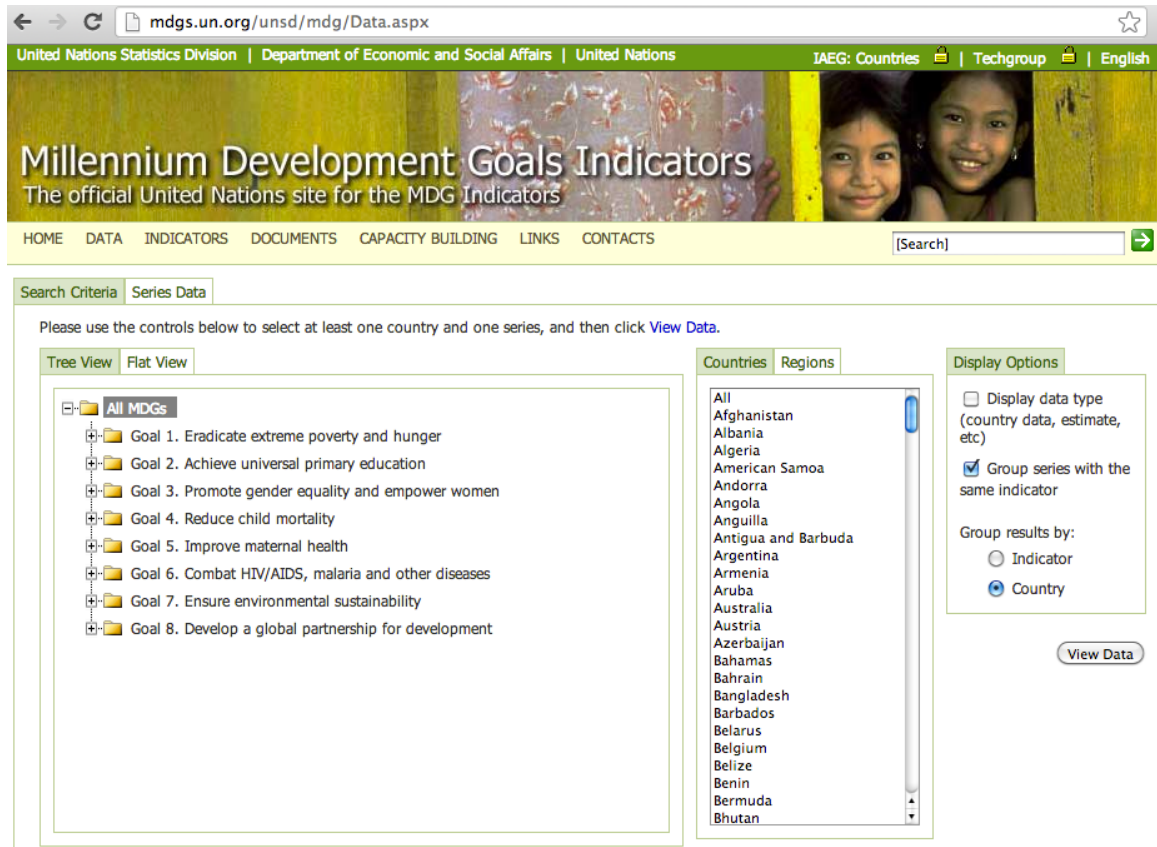


Figure 1.1. The Web-based interface provided for navigating the United Nations Millennium Development Goals Indicator data sets [111]. This is one example of the variety of formats and protocols used for making data available on the Web.

The screenshot shows the Central Intelligence Agency (CIA) website's Library section, specifically the 'The World Factbook' page. The browser address bar indicates the URL: <https://www.cia.gov/library/publications/the-world-factbook/rankorder/2003rank.html>. The page features the CIA logo and the tagline 'THE WORK OF A NATION. THE CENTER OF INTELLIGENCE.'.

The main navigation bar includes links to HOME, ABOUT CIA, CAREERS & INTERNSHIPS, OFFICES OF CIA, NEWS & INFORMATION, LIBRARY, and KIDS' ZONE. The LIBRARY section is active, displaying a sidebar with various publication categories and a main content area titled 'THE WORLD FACTBOOK'.

The main content area shows a 'COUNTRY COMPARISON :: GDP - REAL GROWTH RATE' table. A dropdown menu at the top of the table allows users to 'Please select a country to view'. Below the table, there is a 'DOWNLOAD DATA' link and a 'VIEW TEXT/LOW BANDWIDTH VERSION' link.

RANK	COUNTRY	(%)	DATE OF INFORMATION
1	Libya	104.50	2012 est.
2	Sierra Leone	15.20	2012 est.
3	Afghanistan	12.50	2012 est.

Figure 1.2. The Web-based interface for downloading data from the CIA World Factbook [3]. A data download link is provided that yields a text file using a nonstandard table format. This is a second example of the variety of formats and protocols used for making data available on the Web.

Data in Gapminder World

List of indicators [About countries & territories](#) [Documentation](#) [Data blog](#)

The table below lists all indicators displayed in Gapminder World. Click the name of the indicator or the data provider to access information about the indicator and a link to the data provider.
Indicators labeled "Various sources" are compiled by Gapminder. They can be reused freely but please attribute Gapminder.

List of indicators in Gapminder World

Show Indicators Search:

Indicator name	Data provider	Category	Subcategory	Download	View	Visualize
Adults with HIV (% , age 15-49)	Based on UNAIDS	Health	HIV			
Age at 1st marriage (women)	Various sources	Population				
Aged 15+ employment rate (%)	International Labour Organization	Work	Employment rate			
Aged 15+ labour force participation rate (%)	International Labour Organization	Work	Labour force participation			

Ask a question

Figure 1.3. The Web-based interface for downloading data harvested by the Gap-Minder project [58]. A data download link is provided for each indicator that yields an Excel spreadsheet hosted using Google Docs.

Sometimes visualization interfaces are provided for data published online, such as in figure 1.5, but these tools are typically extremely limited in scope and hard-coded to the data set at hand. With the tools available today such as D3.js, creation of Web-based interactive data visualizations involves hard coding one-off projects to a particular data set. Ideally, anyone should be able to apply interactive visualization techniques to data easily. This dissertation focuses on the challenges in making this a reality and offers a solution based on the data cube concept. The proposed framework reduces the effort required to create Web-based data visualizations by linking reusable visualization components with data sets within the proposed data representation framework, which is based on the data cube model.

Public data tends to be particularly well suited to the data cube model because it typically contains measures about people (or byproducts of human activities) dis-

The screenshot shows a web browser window with the URL 205.207.175.93/Vitalstats/TableViewer/tableView.aspx. The page header includes links to CDC Home, NCHS Home, Contact NCHS, NVSS Home, VitalStats Home, Privacy Policy, and Accessibility. The main header features the CDC logo, the text 'National Vital Statistics System', and 'U.S. DEPARTMENT OF HEALTH AND HUMAN SERVICES'. A 'VITAL STATS' logo is also present. Below the header, there are tabs for 'Tables', 'Table', and 'Chart'. The 'Table' tab is selected. The interface shows a selected demographic characteristic: 'United States, 1993-2010'. A dropdown menu for 'Other:' is set to 'Measure (%) - Births to mothers under 20 years'. The main data table has columns for 'Race/Origin' and 'Year' as row headers, and data columns for 'Total', 'White total', 'Non-Hispanic white', 'Black total', 'Non-Hispanic black', 'American Indian', 'Asian or Pacific Islander', and 'Hispanic'. The data rows span from 1993 to 2010.

Race/Origin	Total	White total	Non-Hispanic white	Black total	Non-Hispanic black	American Indian	Asian or Pacific Islander	Hispanic
Year								
1993	12.8	0.0	9.5	22.7	22.9	20.3	5.7	17.4
1994	13.1	11.3	9.7	23.2	23.3	21.0	5.7	17.8
1995	13.1	11.5	9.8	23.1	23.3	21.4	5.6	17.9
1996	12.9	11.3	9.7	22.8	23.0	20.9	5.3	17.4
1997	12.7	11.2	9.5	22.2	22.4	20.8	5.2	17.0
1998	12.5	11.1	9.4	21.5	21.6	20.9	5.4	16.9
1999	12.3	10.9	9.2	20.7	20.7	20.2	5.1	16.7
2000	11.8	10.6	8.7	19.7	19.8	19.7	4.5	16.2
2001	11.3	10.2	8.2	18.9	18.9	19.3	4.3	15.6
2002	10.8	9.8	7.9	18.0	18.1	18.5	3.8	14.9
2003	10.3	9.4	7.5	17.3	17.4	18.2	3.5	14.3
2004	10.3	9.3	7.4	17.1	17.3	17.9	3.4	14.3
2005	10.2	9.3	7.3	16.9	17.0	17.7	3.3	14.1
2006	10.4	9.4	7.4	17.0	17.2	17.6	3.3	14.3
2007	10.5	9.5	7.5	17.2	17.3	18.4	3.1	14.2
2008	10.4	9.5	7.5	17.0	17.1	18.0	3.0	14.1
2009	10.0	9.2	7.3	16.4	16.4	17.3	2.8	13.8
2010	9.3	8.5	6.7	15.2	15.2	16.1	2.6	13.1

Figure 1.4. The Web-based pivot table user interface for downloading data from the US Centers for Disease Control about births to mothers under age 20 by demographic and year [54]. The product powering this interface is the Beyond 20/20 Web Data Server [1]. In this interface a “download” button is provided that yields data in CSV (Comma Separated Value) format.

tributed across time, space (geographic regions), and other dimensions such as gender or age range. For example, the data available in the GapMinder visualization tool contains measures (such as “number of adults with HIV/AIDS” and “child mortality”) aggregated across countries and years [58]. This partitioning of space into countries and time into years is one choice of levels in the space and time hierarchies, but the data cube model is more general in that it can support multiple levels of detail in both the Space dimension (e.g., Continents, Countries, States, Counties, and Metropolitan Areas) and the Time dimension (e.g., millennia, centuries, years,

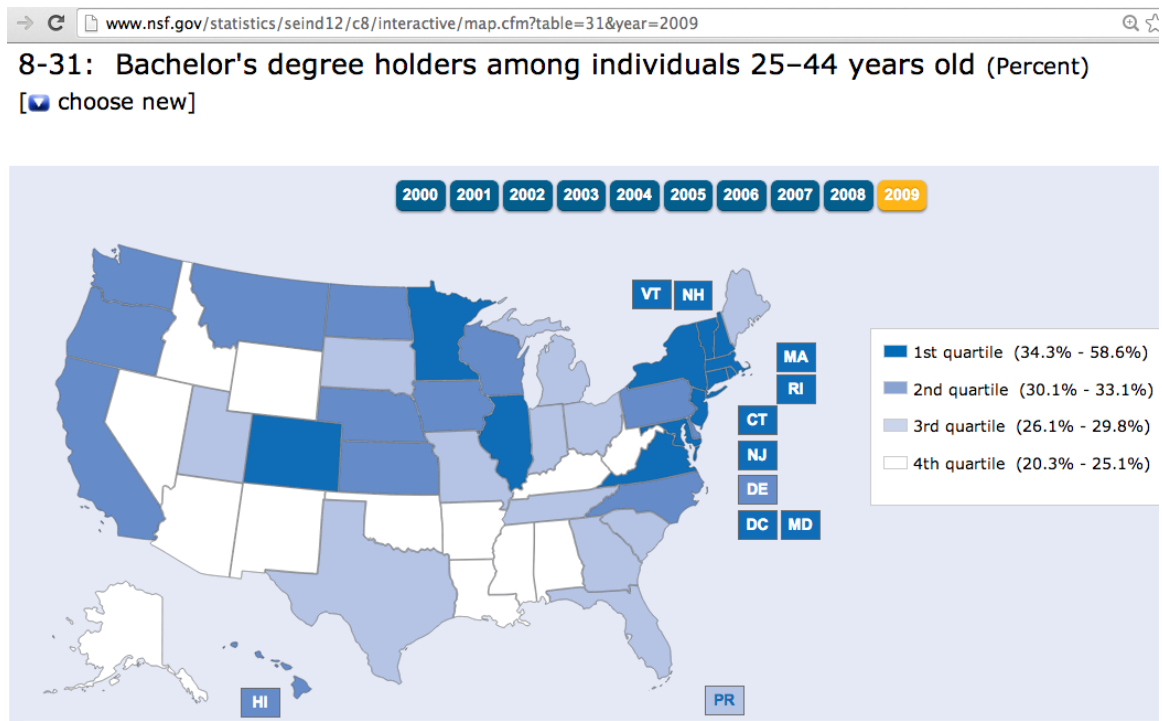


Figure 1.5. An interactive visualization of bachelors degree statistics provided by the National Science Foundation site [55]. This is an example of an extremely limited visualization tool provided along with a data set.

months, days, hours and minutes). Therefore, any data sets that contain measures (also called statistics, indicators, or metrics) aggregated along any resolution of time and space can be modeled as data cubes.

When multiple data sets are modeled as data cubes, they can be integrated into a single structure. Based on the common dimensions and measures shared between data sets, an integrated heterogeneous data cube structure can be created from an arbitrary number of data sets from multiple sources. Interactive visualization techniques can be applied to this integrated structure, yielding fundamentally new ways of exploring and presenting multiple data sets.

To motivate research in data cube integration and visualization, one must consider a larger picture. The data available today can paint a vivid picture of the world if it is exposed in a meaningful way. Data visualization augments human cognition by enabling users to glean knowledge from data using visual perception rather than detailed mental analysis [25]. Data cubes provide a well structured, common representation that captures the essence of many data sets. Data visualization augments human cognition by offloading data analysis tasks to tasks of visual perception. The synthesis of data with visualization through data cubes can lead to a technology platform that changes the world by bringing the power of data visualization to the public.

1.2 Challenges

The main problem this work addresses is the gap between heterogeneous data sets and information visualization software. The reality of the current data visualization landscape contains many disparate data sets, data formats, visualization tools (specific implementations), and visualization techniques (abstract conceptual visualization approaches). The problem with this situation is that it requires an immense amount of manual work to establish a complete pipeline from any given data source

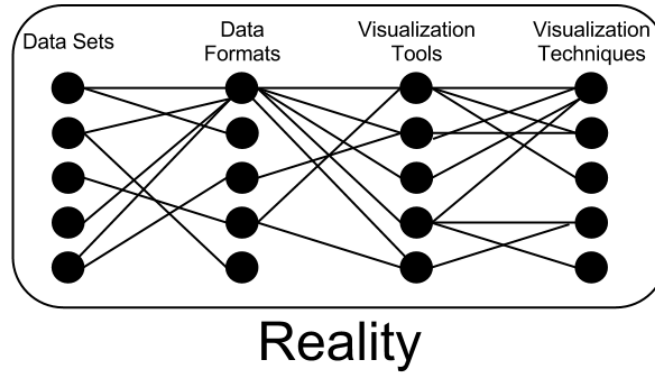


Figure 1.6. The fragmentation of data and visualization.

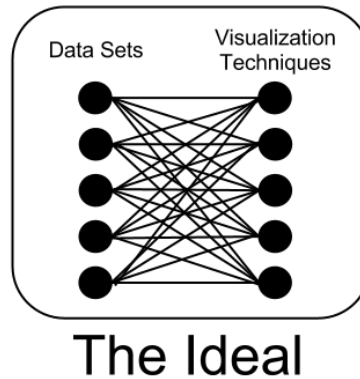


Figure 1.7. Bridging the gulf between data and visualizations.

to an instantiation of a visualization technique. This situation is summarized in figure 1.6.

An ideal solution to this problem would allow any target data set to be visualized using any target visualization technique. For example, the task “Visualize the US Census Population Statistics on a Choropleth Map” should be possible to execute in a straightforward way, ideally by a simple process in which the target data set is selected (US Census Population Statistics), the target visualization technique is selected (Choropleth Map), and the mapping from the data set to the visualization technique is configured (total population maps to region color by a selected color scale). This ideal is summarized in figure 1.7.

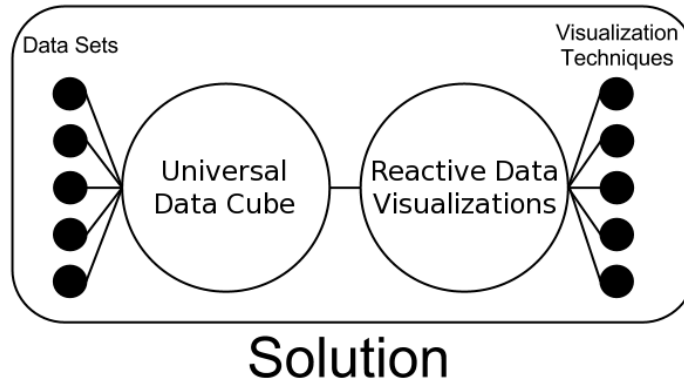


Figure 1.8. Our proposed solution; introduce generic intermediate representations conducive to data visualization.

The proposed solution to address the gulf between data sets and visualization techniques involves the introduction of a generic and powerful intermediate data representation. The generic data representation should be capable of representing most data sets. For this the data cube concept was chosen as a foundation, as it captures the essential structure of many data sets. This solution is summarized in figure 1.8.

1.3 Application Areas

Imagine what it would be like if any person could readily access or construct interactive visualizations of data. Interactive data visualization has relevance to many application areas including, but not limited to, education, journalism, and big data analytics.

Educational material is ripe with opportunity for augmentation by interactive visualizations. For example, consider the next generation of textbooks as eBooks running on tablets. Textbooks covering historical trends can use visualization to represent data about, for example, the distribution of various demographics across the Earth and how they have shifted over centuries. Economics courses could use interactive visualizations of global economic data to help students better understand socioeconomic dynamics. Environmental studies can include visualizations of data

on climate and pollution. Medical studies can take advantage of public health data. There is no end to the potential applications of public data visualization in education.

Journalism requires an in-depth understanding of stories as they evolve. Public data can provide context for those stories, and interactive visualizations of relevant data can be placed in digital publications alongside article text. This places the power of interactive visualization in the hands of readers. Visualizations are already being used for this purpose today by publishers such as the New York Times and the Boston Globe [124].

Big data analytics is a fast growing area of active research and development. The proposed data modeling and visualization approach has relevance for big data analytics because the data cube construct is a particularly well suited structure for presenting summary queries executed across large distributed data stores. For example, Facebook developed a visualization system based on data cubes and interactive specification of slices [138]. In this system, a distributed in-memory database developed at Facebook is interactively queried based on user-defined filters that compute data cube aggregates of real-time data on demand. LinkedIn is taking a similar approach to present real-time aggregated data to users through visualizations [136].

CHAPTER 2

REACTIVE VISUALIZATIONS

Managing complex data flows and update patterns is one of the most difficult challenges in interactive data visualization. Constructing interactive visualizations with multiple linked views can be a daunting task. Functional reactive programming provides approaches for specifying data dependency graphs declaratively and maintaining them automatically. Functional reactive programming can be combined with the Model View Controller (MVC) paradigm to provide reactive models, an effective abstraction that supports construction of complex interactive data visualization systems.

Even with the wealth of visualization toolkits and libraries that exist today, there is a need for an abstraction that addresses the core issue of managing complex data flows and update propagation patterns. In this chapter, a novel approach for developing reusable interactive visualization components using reactive models (reactive visualizations) is introduced. A pseudocode implementation of reactive models is presented in appendix A. A JavaScript implementation of reactive models has been released as the open source ModelJS project [88]. The effectiveness of the proposed approach is demonstrated in several visualization examples including multiple linked views.

2.1 Related Work

The first attempt at a systematic formalization of data visualization was Jacques Bertin’s “Semiology of Graphics” [13]. In this work, Bertin relates data types to

visual marks and channels in a coherent system that takes visual perception into account. Bertin’s work has influenced many future theoretical underpinnings of visualization, including Leland Wilkinson’s “Grammar of Graphics” [155] and Jock Mackinlay’s APT (A Presentation Tool) system [105], which led to the development of the commercial visualization package Tableau [68].

Interactions within data visualization environments have been well studied. Becker et al. investigated brushing in scatter plots [10]. Shneiderman et al. explored dynamic queries in general and how these operations fit into a larger context of visual information seeking [129]. Ward introduced a visualization system based on multiple linked views with direct manipulation techniques including brushing and linking [150]. Anselin discussed how interactive visualization systems with linked views can be applied to Geographic Information Systems [7]. Yi et al. conducted a thorough survey of existing taxonomies for visualization and interactions and developed a set of generalized classes of interactions for visualization [158].

Much work has been done regarding interactive visualization of data cubes. Stolte et al. introduced a formalism for defining multi-scale visualizations of data cubes throughout their work on the Polaris system [135] [134] [133]. Cuzzocrea et al. surveyed the area of data cube visualization in depth [38]. Mansmann coined the term “Visual OLAP” and framed it as a fundamentally new paradigm for exploring multi-dimensional aggregates [108]. Scotch et al. developed and evaluated SOVAT, a Spatial OLAP visualization and analysis tool applied to community health assessments [128] [127]. Techapichetvanich et al. explored how visualization interactions pertain to data cubes in particular [139]. Sifer et al. introduced a visual interface using coordinated dimension hierarchies for OLAP cubes [132]. Several interactive “Big Data” visualization systems have been introduced that use the data cube structure [98] [100].

Interactions within data visualization environments have been well studied. Becker et al. investigated brushing in scatter plots [10]. Shneiderman et al. explored dy-

dynamic queries in general and how these operations fit into a larger context of visual information seeking [129]. Ward introduced a visualization system based on multiple linked views with direct manipulation techniques including brushing and linking [150]. Anselin discussed how interactive visualization systems with linked views can be applied to Geographic Information Systems [7]. Yi et al. conducted a thorough survey of existing taxonomies for visualization and interactions and developed a set of generalized classes of interactions for visualization [158].

The World Wide Web has evolved to become a full-fledged application development platform. HTML5 is the latest set of standards and Application Programming Interfaces (APIs) from the World Wide Web Consortium (W3C) that define the capabilities of modern Web browsers [73]. HTML5 applications are able to run across multiple platforms (albeit requiring some effort from developers). HTML5 has eclipsed Java Applets and Flash in fulfilling the dream of “write once, run anywhere.” HTML5 contains three graphics technologies that can support interactive Web-based visualizations: the Document Object Model (DOM), Canvas [56], Scalable Vector Graphics (SVG) [42], and WebGL [109].

D3.js is a flexible and powerful visualization library that uses SVG and has a strong community of users [19]. At its core, D3 is a DOM manipulation library with heavy use of functional programming. D3 allows concise declarative statements to define the core logic of visualizations. D3 provides additional APIs for performing common visualization tasks such as defining and using scales, generating labeled axes, and computing layouts from graphs and trees. D3 is at the center of a vibrant developer ecosystem and has seen wide adoption in industry. There are plentiful examples of D3.js usage for creating visualizations [21]. Many supporting libraries have been created including NVD3 reusable charts, Chart.js for composing visualization elements, Crossfilter.js for interactive multidimensional filtering, and DC.js for multiple linked

views. Other reusable chart libraries based on D3 include Dimple, RAW, VEGA, reD3 and Forio Contour.

Interactive data visualizations can be linked together such that interactions in one visualization cause updates in another visualization. This technique is referred to as “multiple linked views” [122] and “brushing and linking” [87, 8]. This technique overcomes limitations of single visualizations by supporting exploration of the data through interaction. More information can be presented to the user with multiple linked views as compared to static visualizations. In fact, interactive linked views represent the same amount of data as small multiples with only a single visible visualization instance. Interaction takes the place of additional screen real estate. Dynamic queries, a technique related to multiple linked views, allow the user to define query parameters interactively. The interactively defined query parameters are used for generating the input data for a visualization [129].

The Model-View-Controller (MVC) architecture is a long standing best practice for organizing complex applications [44]. The MVC architecture was first introduced as part of the Smalltalk-80 system for building user interfaces [92] and has been used extensively for Web application development [94]. Several authors describe how the MVC architecture can be applied to visualizations with multiple linked views [72, 69, 151, 23].

Functional reactive programming provides techniques for declaratively specifying reactive data dependency graphs [149]. Elliott et al. applied functional reactive programming to animation [52]. Hudak et al. applied functional reactive programming to robotics [76]. Data flow is a concept related to functional reactive programming in which developers can specify directed graphs of data transformations [65]. The KNIME data analysis environment uses a data flow model as its primary abstraction [12].

2.2 Reactive Models

Functional reactive programming can be combined with the MVC paradigm to create reactive models. These reactive models can serve as a foundation for reusable interactive visualization components. This approach overcomes limitations of traditional MVC frameworks and is simpler than using a full blown functional reactive programming framework. This section discusses a simple model with only *set* and *get* methods (*SimplestModel*), then a more complex version that also includes *on* (*SimpleModel*), then finally a complete reactive model implementation that includes the *when* operator from functional reactive programming (*Model*).

The Model in MVC paradigm is responsible for:

- managing the state of the application,
- allowing the Controller to change the state of the application, and
- notifying the view when the state of the application changes.

One simple and widely used method for structuring a Model is as a set of key-value pairs [94]. This kind of model can fulfill all the responsibilities of a Model with three methods:

- *set(key, value)* Set the value for a given key.
- *get(key)* Get the value for a given key.
- *on(key, callback)* Add a change listener for a given key. Here, *callback* is a function that will be invoked synchronously when the value for the given key is changed.

The following pseudocode implements a key-value model that has only *set* and *get* methods. Line 1 defines the constructor function, *SimplestModel*, which will return a new object that has *set* and *get* methods. Line 2 defines a private variable *values*

that will contain the key-value mapping. Lines 3 - 5 define the *set* and *get* methods, which store and retrieve values from the internal *values* object. The pseudocode conventions are given in appendix A.

```
1  SimplestModel =  $\lambda()$ 
2    values = { }
3    return
4      set :  $\lambda(key, value)$  values[key] = value
5      get :  $\lambda(key)$  return values[key]
```

Here's an example of how *SimplestModel* might be used.

```
1  mySimplestModel = SimplestModel()
2  mySimplestModel.set('x', 5)
3  mySimplestModel.get('x') // Evaluates to 5
```

Here is a version of the model that implements the *on* method as well:

```

1  SimpleModel =  $\lambda()$ 
2    values = { }
3    callbacks = { }
4    return
5      on :  $\lambda(\textit{key}, \textit{callback})$ 
6        if callbacks[key] == NIL
7          callbacks[key] = []
8          callbacks[key].push(callback)
9      set :  $\lambda(\textit{key}, \textit{value})$ 
10        values[key] = value
11        if callbacks[key]  $\neq$  NIL
12          for callback  $\in$  callbacks[key]
13            callback()
14      get :  $\lambda(\textit{key})$  return values[key]

```

The above version includes an additional private variable, *callbacks*, which is an object whose keys are property names and whose values are arrays of callback functions. The *on* method defined starting at line 5 adds the given callback to the list of callbacks for the given key (and creates the list if it does not yet exist). The *set* method has been modified to invoke the callback functions associated with the given key when the value for that key is changed.

Here is an example of how the *on* method can be used.

```

1  mySimpleModel = SimpleModel()
2  mySimpleModel.on('x',  $\lambda()$ 
3    log(mySimpleModel.get('x'))
4  )
5  mySimpleModel.set('x', 5) // Causes line 3 to log 5
6  mySimpleModel.set('x', 6) // Causes line 3 to log 6

```

For complex applications such as interactive visualizations, managing propagation of changes can quickly become complex. For this reason, modular visualization environments based on data flow have become popular [2]. A data flow graph defines a directed acyclic graph of data dependencies. The data flow model is amenable to construction of graph-based visual programming languages [74]. While many systems consider data flow as a means to construct data transformation pipelines, the concept also applies to building reactive systems that manage change propagation throughout an application or subsystem in response to user interactions or other events [52].

To provide a solid foundation for dynamic visualization systems, the Model should be able function in the context of data dependency graphs. Developers should be able to specify data dependencies declaratively, and change propagation should be managed automatically. The *when* operator from functional reactive programming propagates changes from one or more reactive functions (such as is found in the JavaScript libraries Bacon.js [32] and RXJS [34]).

The SimpleModel implementation can be extended with a *when* operator that enables construction of data dependency graphs. This operator will become a foundation for building dynamic interactive visualizations. Since *when* is superior to *on* in that it handles change propagation intelligently, in this final version *on* is not exposed in the public Model API. Adding *when* depends on having some utility functions available, *debounce* and *allAreDefined*.

```

1  debounce =  $\lambda$ (callback)
2      queued = FALSE
3      return  $\lambda$ ()
4      if queued == FALSE
5          queued = TRUE
6          run( $\lambda$ ())
7          queued = FALSE
8          callback()
9      )

```

The *debounce*(*callback*) function returns a function that, when invoked one or more times in a single code path, will queue the given *callback* function to execute only once on the next tick of the event loop. This has the effect of collapsing multiple sequential calls into a single call. The returned function is referred to as the “debounced” function.

The *debounce* function defined starting on line 1 creates a closure with a boolean variable *queued* (instantiated on line 2) that keeps track of whether or not the *callback* function is currently queued to execute in the future. When the debounced function (defined starting on line 3) is called the first time, the condition on line 4 evaluates to TRUE. This causes *queued* to be set to TRUE (on line 5) and also causes the function defined starting on line 6 to be queued to run in the future. This uses the built-in function *run* that queues a function to execute on the next tick of the event loop.

When the debounced function is invoked multiple times in the same code path, the condition on line 4 evaluates to FALSE, and nothing happens. When the current code path terminates and the queued function is invoked, *queued* is set to FALSE (on line 7) and the *callback* function is invoked.

The function *allAreDefined*(*array*) checks if all values in the given *array* are defined. It does so by comparing each item in the array to the special value NIL. As

soon as one item is found to be NIL, the function returns FALSE (on line 4). If all items have been checked and none are found to be NIL, the function returns TRUE (on line 5).

```

1  allAreDefined =  $\lambda(array)$ 
2    for item  $\in$  array
3      if item == NIL
4        return FALSE
5    return TRUE

```

We are now ready to define our model that includes the *when* operator.

```

1  Model =  $\lambda()$ 
2    simpleModel = SimpleModel()
3    return
4      set : simpleModel.set
5      get : simpleModel.get
6      when :  $\lambda(dependencies, fn)$ 
7        callFn = debounce( $\lambda()$ 
8          args = dependencies.map(simpleModel.get)
9          if allAreDefined(args)
10             apply(fn, args)
11        )
12      callFn()
13      for key  $\in$  dependencies
14        simpleModel.on(key, callFn)

```

The above *Model* pseudocode implements a reactive model. Line 2 instantiates a *SimpleModel* instance that serves as the core of the reactive model. The *set* and *get* methods of the inner *SimpleModel* are exposed in the reactive model instance. Note,

however, that the *on* method is not exposed. The *when* method defined starting on line 6 implements reactive data flow propagation. This method takes as input two arguments, a *dependencies* array of model property names, and a callback called *fn*. Line 7 defines *callFn*, a debounced function that invokes the *fn* callback.

The callback function *fn* gets invoked when all dependency properties are available and whenever any dependency properties change. The values for each dependency property are extracted from the model on line 8 and passed as arguments to the callback function on line 10. Line 9 ensures that callbacks are only invoked if all dependency properties have assigned values. Line 12 invokes the callback once for initialization. The *callFn* function is added as a listener to all dependency properties using the *on* method on lines 13 and 14. Note that when the *fn* callback sets model properties, this represents edges in the data flow graph from each of the dependency properties to the newly set values. This implementation uses the JavaScript event loop (by debouncing) as a queue to perform breadth-first update propagation through reactive data flow graphs.

The following pseudocode demonstrates basic usage of reactive models in computing a full name from first and last names. In this example, line 1 instantiates a new reactive model and assigns it to the variable *person*. Line 2 invokes the *when* method with dependency properties *firstName* and *lastName*. When both properties are defined and whenever either one changes, the callback function implemented on line 3 gets invoked. This callback function sets the *fullName* property on the *person* model to be the full name, that is, the first and last names combined with a space between them. This reactive flow is depicted in figure 2.1.

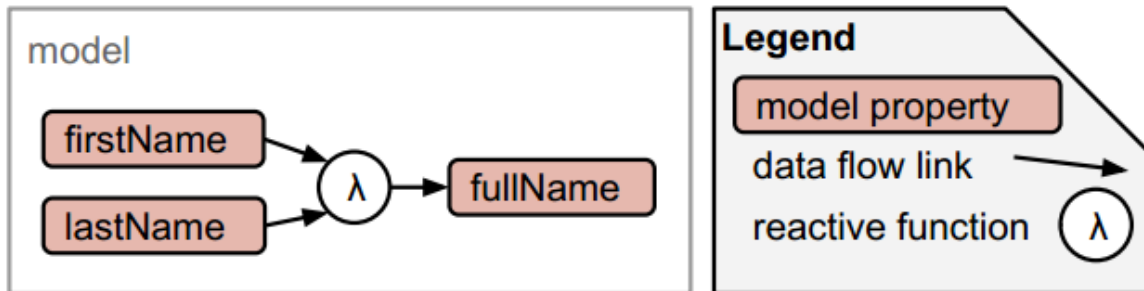


Figure 2.1. A basic reactive model that computes the full name whenever either the first name or last name changes. The reactive function is defined using the *when* operator, therefore represents a node in a reactive data flow graph.

```

1  person = Model()
2  person.when(["firstName", "lastName"],  $\lambda$ (firstName, lastName)
3    person.fullName = firstName + " " + lastName
4  person.when(["fullName"],  $\lambda$ (fullName)
5    log(fullName)
6  person.set("firstName", "John")
7  person.set("lastName", "Smith")

```

Lines 4 and 5 set a *when* callback to log the full name whenever it changes. Lines 6 and 7 set the *firstName* and *lastName* properties on the *person* model. This causes *fullName* to be computed and set on the model. When *fullName* is set, the callback that logs *fullName* is invoked, causing the string "John Smith" to be logged.

2.3 Reactive Bar Chart

Reactive models can serve as a foundation for interactive visualizations. Reactive models for multiple visualizations can be linked together at a higher level to form linked views. Interactive visualizations must respond to changes made by users such as resizing the display, changes in the data driving the visualization, changes

in visualization configuration, and updates from other visualizations in a linked view context. This section introduces reactive visualization concepts through the example of a Bar Chart, then presents reusable reactive visualization components extracted from the Bar Chart.

A bar chart takes as input an array of data entries and a configuration that specifies the mapping from data attributes to the X and Y axes. It yields a dynamic bar chart graphic as output. The behavior desired for building the bar chart in response to changes in data and configuration fit perfectly within the framework of reactive models. As a first pass at constructing a reactive bar chart, the following reactive model properties are introduced:

- **data** The input data table
- **xAttribute** The attribute used for the X scale (bar name)
- **yAttribute** The attribute used for the Y scale (bar height)

Using these three properties alone supports the essence of a bar chart, the plotting of bars and the labeling of axis tick marks. However, as attribute names are often cryptic and may not be the best labels for visualizations, two more model properties can be introduced that specify the text content of X and Y axis labels:

- **xAxisLabel** The string displayed as the X axis label
- **yAxisLabel** The string displayed as the Y axis label

Bar charts and many other visualizations have an inner visualization rectangle in which visual marks are plotted. This inner rectangle lies within the outer rectangle containing the entire visualization, offset from the outer box by a specified margin. To integrate margin logic with reactive models, the following model properties are introduced:

- **size** The size of the outer rectangle that contains the entire visualization, defined as an object with *width* and *height* properties in pixels.
- **margin** The margin object, having properties **left**, **right**, **top** and **bottom** in pixels, according to the D3 margin conventions [20].
- **width** The width of the inner visualization rectangle in pixels.
- **height** The height of the inner visualization rectangle in pixels.

The model properties defined this far support encapsulation of conventional D3 margins. A reactive function can be defined that updates *width* and *height* based on *size* and *margin*. With the attributes present, scales can also be computed by reactive functions. The X scale depends on *data*, *xAttribute*, and *width*. The Y scale depends on *data*, *xAttribute*, and *height*. With the X and Y scales defined, the last remaining step is to compute the bars and axes from the data and scales. The complete reactive flow graph for a bar chart is shown in figure 2.2.

2.4 Reusable Reactive Flows

Consider the following visualization techniques:

- Bar Chart
- Scatter Plot
- Line Chart
- Stacked Area Chart

These visualizations share many underlying primitives such as scales, axes, and margins. The D3 Open Source project provides high quality generalized solutions for these and many more visualization primitives [19]. These visualization primitives

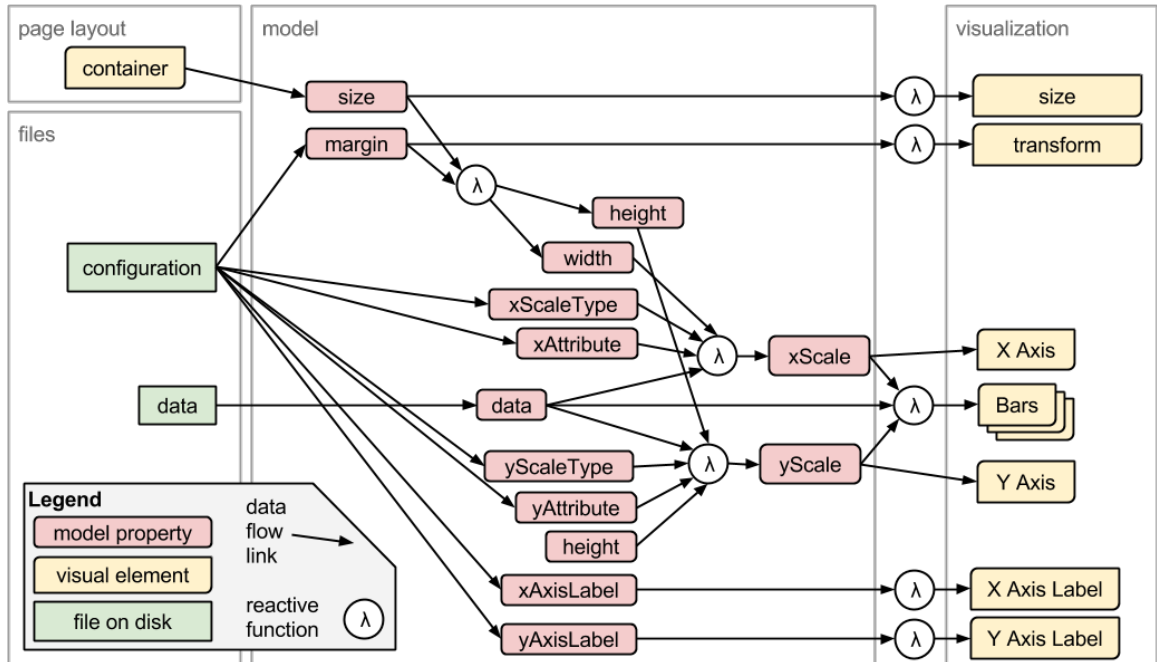


Figure 2.2. The data flow graph for a reactive bar chart based on *reactive models*.

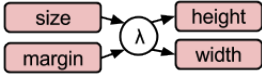
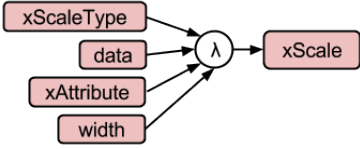

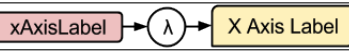
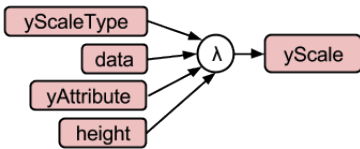


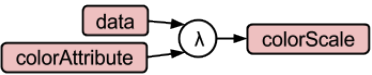
and their computational dependency structure can be encapsulated by reactive flow graphs. Interactive forms of these visualizations also share interaction techniques for selecting visual marks such as rectangular brushing, hovering, clicking, panning, and zooming. Table 2.1 lists reusable reactive flows common to many visualizations.

Other visualization techniques that may also be implemented using reactive models as a foundation include:

- Parallel Coordinates
- Choropleth Map
- Table
- Box Plot

Property names serve as the common elements between components. Property names used in one or more components not introduced in the Bar Chart include:

Table 2.1. Reusable flows for reactive visualization.

Component	Diagram	Description
margin		Computes the size of the inner visualization rectangle based on the container size (which may change when the user resizes the visualization) and the configured margin.
xScale		Computes the X scale. The domain is computed from the input data by evaluating the X attribute bounds. The range is computed from the inner visualization width.
xAxis		Renders the X Axis (line, tick marks and labels) from the X scale .
xAxisLabel		Renders the text label for the X Axis .
yScale		Computes the Y scale. The domain is computed from the input data by evaluating the Y attribute bounds. The range is computed from the inner visualization width.
yAxis		Renders the Y Axis (line, tick marks and labels) from the Y scale .
yAxisLabel		Renders the text label for the Y Axis .
colorScale		Computes the color scale. The domain is computed from the input data by evaluating the set of unique values found in the color attribute.

- **xScale** The scale (domain and range) for the X axis.
- **xScaleType** **xScale** qualifier: linear, logarithmic, or ordinal.
- **xAxis** The visible X axis (tick marks and labels).
- **yScale** The scale (domain and range) for the Y axis
- **yScaleType** **yScale** qualifier: linear, logarithmic, or ordinal.
- **yAxis** The visible Y axis (tick marks and labels).
- **colorAttribute** The scale used to determine color of visual marks.
- **colorScale** The scale used to determine color of marks.

Table 2.1 lists components that can be combined to easily generate a foundation for a variety of interactive visualizations. These components encapsulate reusable reactive flows that implement the primitives necessary for interactive visualizations. Figure 2.2 showed how several of these components can be assembled to create a general-purpose reactive bar chart. This bar chart flow represents a template for other visualizations, such as scatter plots. In fact, the only things that need to be changed in the bar chart flow to make it a scatter plot are (1) the X axis must be made quantitative, and (2) dots should be plotted rather than bars. Similarly, only the visual marks must be modified to change a scatter plot to a line chart, shown in figure 2.3.

Interactive user interface components can be linked with any reactive model property. This makes it straightforward to add interactivity to reactive visualizations using conventional user interface elements such as dropdown menus, check boxes, sliders, and color pickers. For example, the X and Y attributes used by the scatter plot in figure 2.4 can be made configurable using dropdown menus.

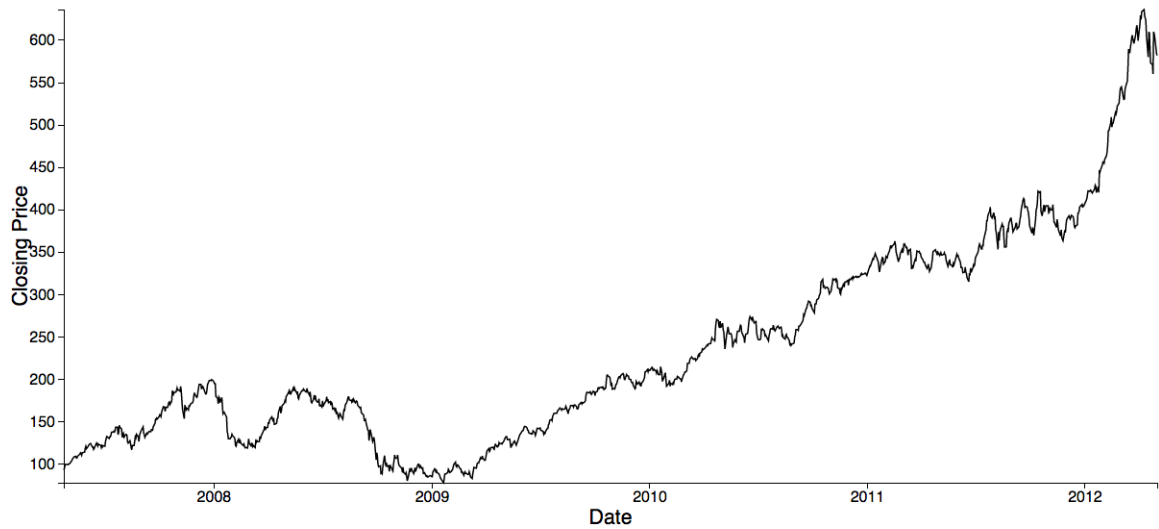


Figure 2.3. A line chart that shares scale and axis flows with scatter plots and bar charts.

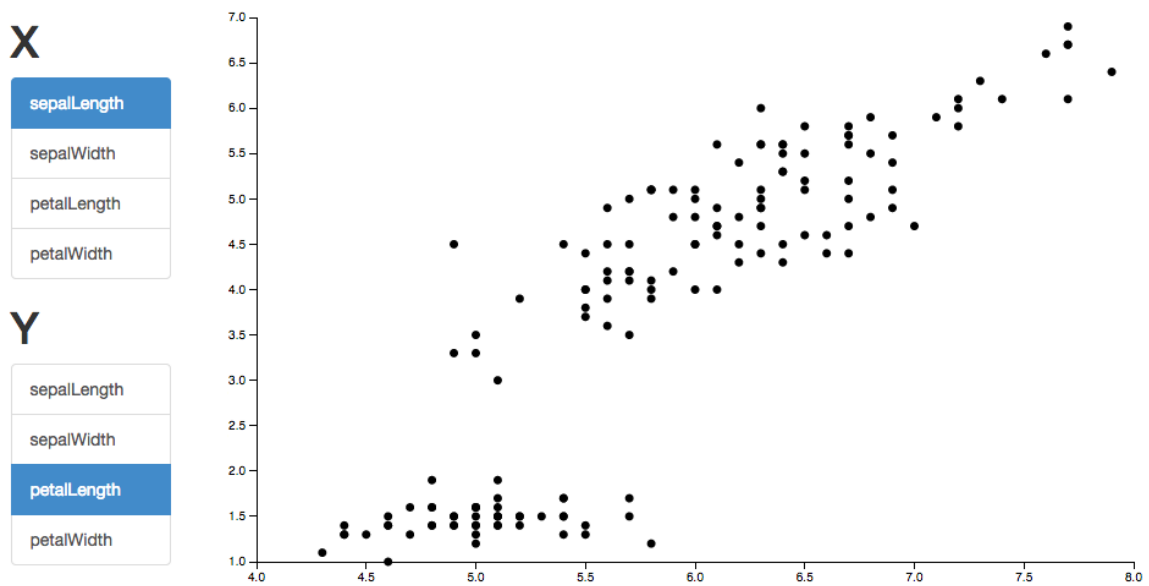


Figure 2.4. A scatter plot where X and Y attributes are selectable via Bootstrap List Group user interface elements.

2.5 Linked Views

Composing reactive visualizations from reusable flows yields reusable and composable interactive visualizations. Interactions such as brushing can be linked to the reactive model of the visualization instance. In the case of brushing, a model property `brushedIntervals` contains the currently brushed intervals. This object stores the (min, max) values for each attribute brushed. The updating brushed intervals can be used to as input to a filter operation that excludes data entries outside the brushed intervals. The output from the filter operator can be routed as input to another visualization. Figure 2.5 shows an example of linked views using this approach.

Figure 2.6 shows the overall flow of the linked scatter plot and bar chart. The brushing interaction sets a property on the reactive scatter plot called `selectedData`. A reactive function that aggregates the selected data by Iris species links the selected data to the input data of the bar chart. Whenever the user brushes to select a new set of records in the scatter plot, the bar chart updates immediately to show only the selected data.

One advantage of reusable reactive flows is that when improvements are be made to a generalized feature, many visualizations manifest the improvement. For example, consider the X and Y axes used for many visualizations such as scatter plot and bar chart. The original D3 example that was drawn from to implement the axes had placed the axis labels inside the inner visualization rectangle. This had the unfortunate consequence that sometimes the label was occluded by marks within the visualization. To solve this issue, the axes were modified such that labels are placed outside the visualization area, are larger than tick mark labels, and are centered with respect to the axes. Figure 2.7 shows the effect of the axis label improvement, which appears both in the scatter plot and bar chart.

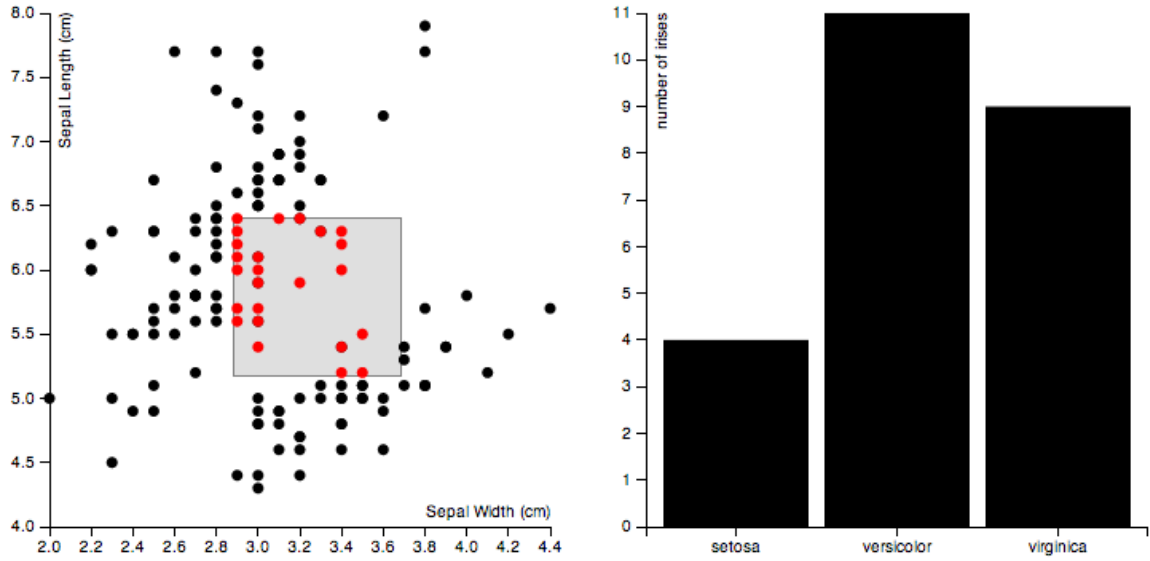


Figure 2.5. A visualization of the Iris data set [6] using linked views, powered by reactive visualization components. Brushing to select records in the scatter plot causes the selected data to be aggregated and displayed in the bar chart.

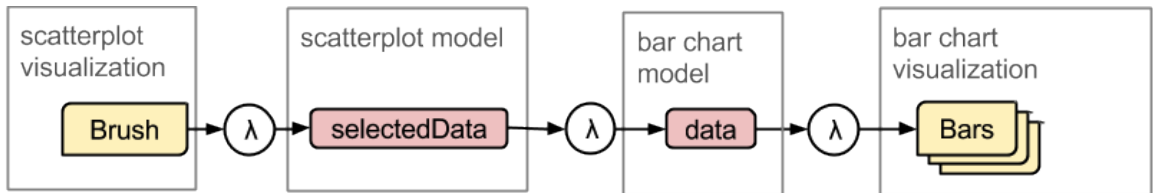


Figure 2.6. The (simplified) data flow graph for the linked scatter plot and bar chart example shown in figure 2.5.

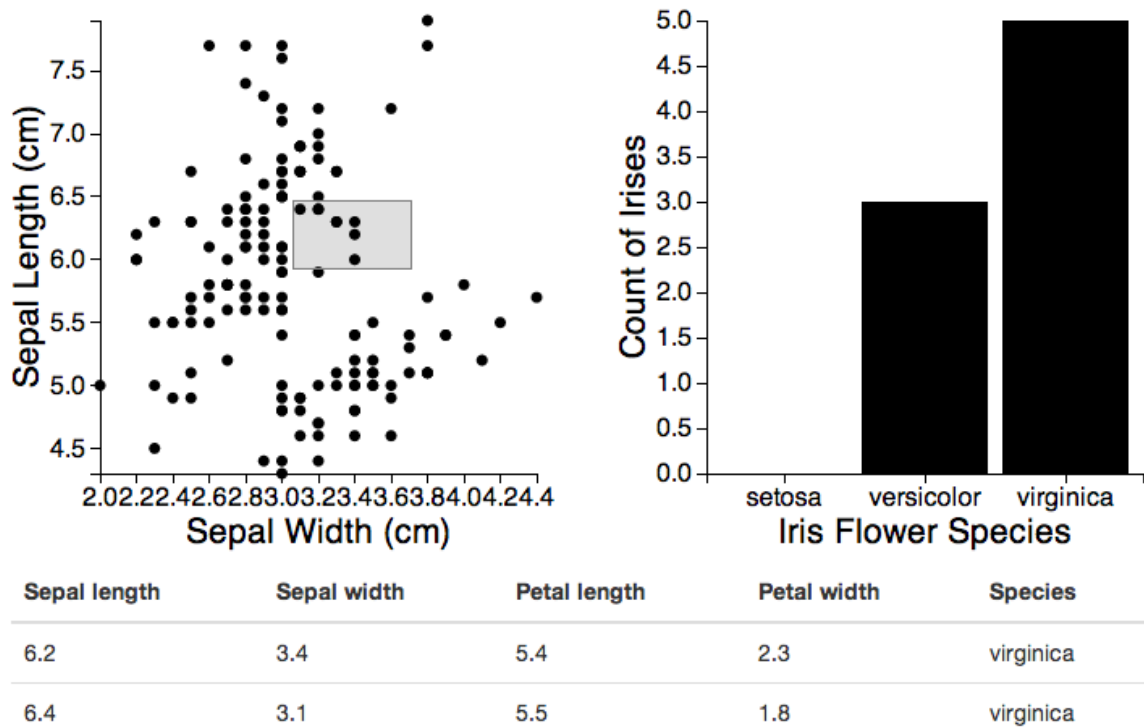


Figure 2.7. An improved version of the linked bar chart and scatter plot example. When changes are made in reusable components such as axes, the improvement is inherited by many reusable visualizations. This version also integrates with the reusable Bootstrap table component, showing the brushed selection in the table.

2.6 Summary

This chapter introduced a novel way to combine elements of functional reactive programming with the Model View Controller (MVC) paradigm to create reactive models. Reactive models allow developers to specify data dependency graphs declaratively. This kind of abstraction is well suited for developing interactive visualizations because it drastically simplifies management of complex data flows and update patterns.

A collection of reusable reactive flows for interactive visualization was also introduced. These reactive flows encapsulate visualization primitives such as margins, scales, and axes. These flows were used as building blocks to generate a reusable bar chart, scatter plot and timeline. Visualizations with multiple linked views can be developed from these reusable visualization components in a straightforward manner.

2.7 Future Work

Future directions for this work will focus on developing a full catalog of reusable visualization components, coupling the data to currently available public data sources, visualization-centric user interfaces, and collaboration.

So far, the reactive visualization approach has been applied only to several visualization techniques. However, the following visualization techniques can also be supported:

- Color Legend
- Pie Chart
- Choropleth Map
- Parallel Coordinates

- Heatmap
- Stacked Bar Chart
- Stacked Area Chart
- Streamgraph
- TreeMap
- Force Directed Graph Layout

Additionally, reusable components for the following interaction techniques can be encapsulated independently of any specific visualization technique using reactive models:

- Brushing - Dragging to define a selection region interactively.
- Picking - Selecting a single visual mark by clicking or tapping on it.
- Details-on-demand - A pattern of linked view composition in which an overview visualization can be used as navigation for a detailed view. This is a fundamental concept in interactive information visualization [130].

A user interface for quickly assembling reusable components together based on graph drawing can also be developed. This user interface would show the data dependency diagrams similar to figure 2.2, but they would be dynamic and editable. Another direction for future research is integrating external user interface components such as traditional list selections, drop down menus, and radio buttons. One example of this concept is shown in figure 2.8, which shows how reactive models can provide reactive HTML tables that can be linked with interactive visualizations. This would help in constructing intuitive user interfaces for manipulating the configuration of visualizations.

Name	Economy (mpg)	Displacement (cc)	Power (hp)	Weight (lb)
AMC Ambassador Brougham	13	360	175	3821
AMC Matador	15	258	110	3730
AMC Pacer	19	232	90	3211
Audi 5000	20.3	131	103	2830
Buick Century Limited	25	181	110	2945
Buick Century Luxus (Wagon)	13	350	150	4699
Buick Lesabre Custom	13	350	155	4502
Cadillac Eldorado	23	350	125	3900
Chevrolet Chevette	29	85	52	2035
Chevrolet Impala	14	454	220	4354
Chevrolet Monte Carlo	15	400	150	3761
Chevy C10	13	350	145	4055

Figure 2.8. A visualization of the Cars data set [70] by a reactive component that renders an HTML table style using Twitter Bootstrap [96].

CHAPTER 3

COLLABORATIVE VISUAL DATA EXPLORATION

The utility of interactive visualizations depends on the ability of users to create them. Typically, data scientists or analysts work in teams. Contributions are made from every person on the team, and then the team must present their findings in a non-exploratory context. Sometimes it is useful to be able to show the path of exploration that the team has taken, using some kind of history model. This chapter presents an approach to enabling all of the above features, leveraging reactive models.

3.1 Related Work

Several projects have focused explicitly on visualization of public data on the Web. ManyEyes was an experiment in scaling the audience for visualizations by empowering users to create visualizations of their own data [147]. ManyEyes provided a fixed set of pre-packaged visualization tools and allowed users to visualize their own data tables using the provided visualizations. GapMinder is a project aimed at exposing public data (primarily the United Nations Millenium Development Goals Indicators) using visualization [123]. GapMinder includes an animated scatter plot with an interactive time slider, a line chart showing statistics over time, and a world map (see figure 3.1). The Google Public Data Explorer provides a visual interface to selected public data sets similar to GapMinder, but it does not make the data available to users in machine-readable form [77].

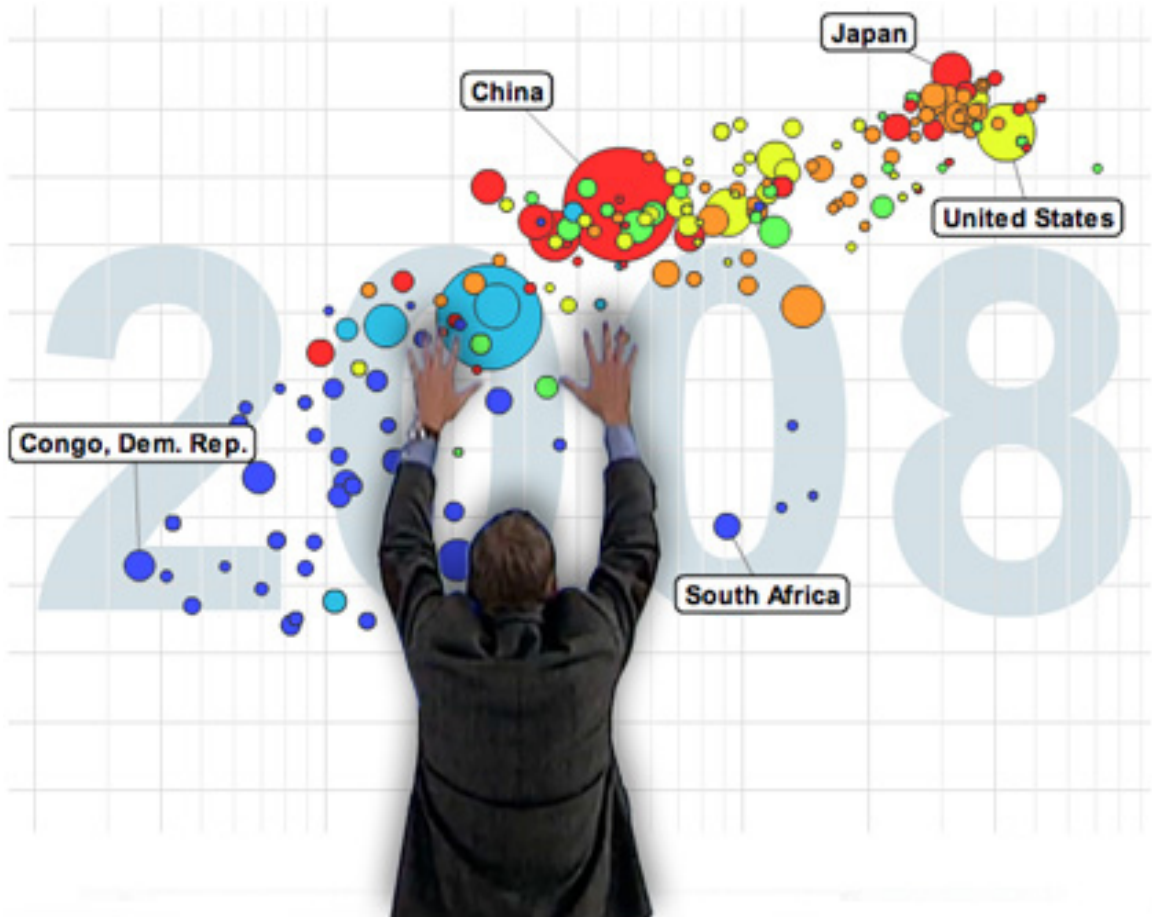


Figure 3.1. GapMinder, a public data visualization tool based on an animated scatter plot, timeline, and map. Here, Professor Hans Rosling, the creator of GapMinder, is shown gesturing the motion of the plot while presenting the visualization.

Many data visualization and analysis tools have been developed with collaboration in mind. Tableau Server has collaboration features such as sharing visualizations, commenting on visualizations, embedding visualizations in Web pages, and sharing filtered data. The OpenChorus project supports annotation and sharing of data sources using a variety of database technologies including SQL, Hadoop (HDFS), Oracle, and Greenplum [93]. Foundational collaboration technologies include Operational Transform for synchronous collaboration such as Google Docs [137] and revision control systems for asynchronous collaboration such as GitHub [117, 41].

3.2 Application State Model

An application typically consists of instantiations of many reusable components. For example, an AngularJS application instantiates reusable components encapsulated as Angular directives, and a BackboneJS application instantiates Views for specific Models.

Interactive visualizations with multiple linked views share common patterns, including layout based on nested boxes and view linkage based on flows between interaction output, data transformations, and visualization input. This configuration can be expressed as a set of reactive models. Some models, such as the layout and linkages, must be able to access other models in the system and change their properties (e.g., the bounding box for layout and the selected elements for linked views). A simple structure based on named models can accommodate the above needs.

Our application state structure consists of *components*, each of which has:

- *alias* The string identifier for the component.
- *module* The string that defines which module to instantiate.
- *model* A collection of serialized model properties.

This structure can be serialized using JSON [37]. The overall application state configuration is an object whose keys are component aliases, and whose values are component objects. Each component object contains key-value pairs representing its serialized model state. Each component also has a value for the `module` property that determines which module is invoked to instantiate the component at runtime. Figure 3.2 shows an example of a configuration that instantiates simple components into a nested box layout. This layout technique is one of the foundations for assembling linked views. Any reusable visualization can be placed into a box in the layout, then linked with other visualizations within the same layout. Figure 3.3 shows an example nested box layout that includes visualizations.

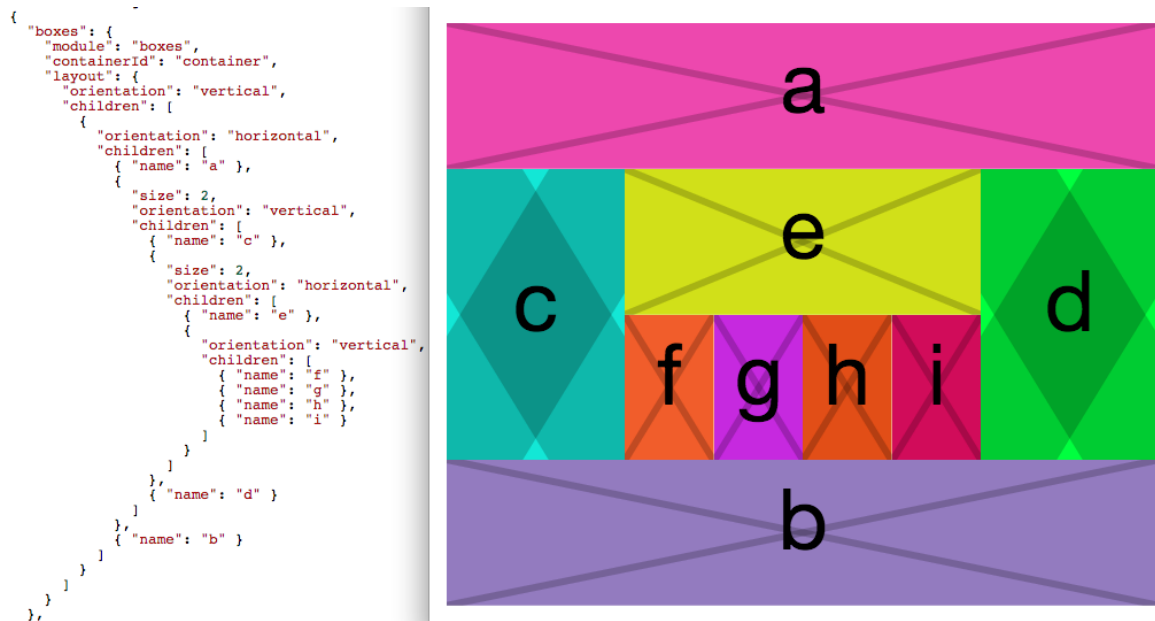


Figure 3.2. An example configuration that defines a nested box layout.

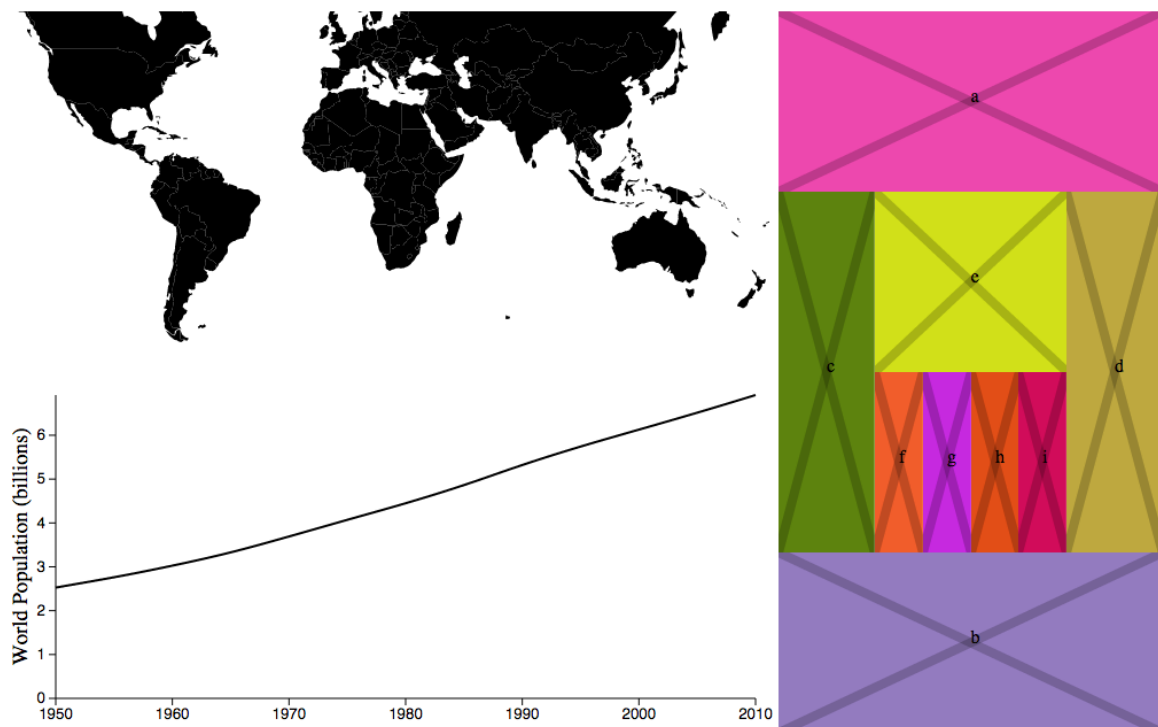


Figure 3.3. An example configuration that defines a nested box layout including line chart and map visualizations.

3.3 Runtime Engine

The application state model addresses the configuration and serialization structure of the application state. A runtime model is necessary to transform serialized application state models into a running system. The runtime engine performs this transformation as follows:

- The runtime engine module loader is instantiated. This process defines a mapping from module names to constructor functions that instantiate the components, returning reactive models.
- The `module` property of each component in the application state configuration is used to instantiate each component.
- The runtime engine maintains a dictionary whose keys are component aliases and whose values are instantiated components.
- Instantiated components can request references to other instantiated components from the runtime. This operation must be asynchronous to account for dynamic module loading.

As an example, consider the flow of instantiation for the application state shown in figure 3.2. The runtime uses the `module` property on each component to fetch and invoke its constructor module. The `boxes` module computes the nested box layout. The computed nested box layout is used for dynamically positioning and sizing the interactive visual components.

The first prototype of the interactively configurable runtime engine was developed during a summer internship at Rapid7, a cybersecurity company, to create an interactive visualization dashboard with multiple linked views for analyzing corporate login activity. Technologies used for visualization included D3.js, a visualization framework that uses SVG, and Leaflet.js, a framework for geographic maps. The map showed

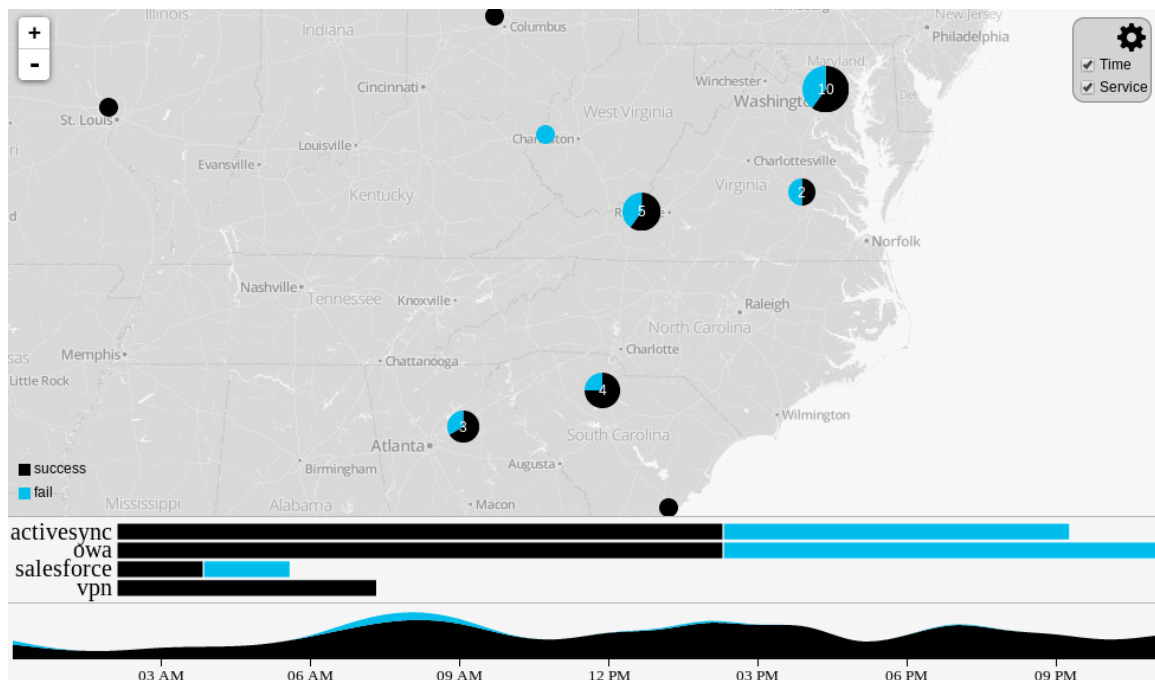


Figure 3.4. This visualization dashboard shows corporate login data and is integrated into the Rapid7 product called UserInsight.

where users have logged into the network, aggregated geographically using the Leaflet MarkerCluster plugin and visualized using D3’s Pie Chart layout. Black represented successful logins and blue represented failed logins. This industry application of our dashboard layout framework demonstrates its capability to define dashboards with multiple linked views. The resulting visualization dashboard is shown in figure 3.5.

3.4 Changing State

When the application state configuration changes, the runtime components must be updated accordingly. Consider the scenario of a user manually editing the application state configuration using a text editor. Every time the user changes the configuration, the system must update the runtime to reflect the new configuration. A naïve approach to this problem is to tear down and re-instantiate the entire runtime whenever the configuration changes. This approach, while functionally correct,

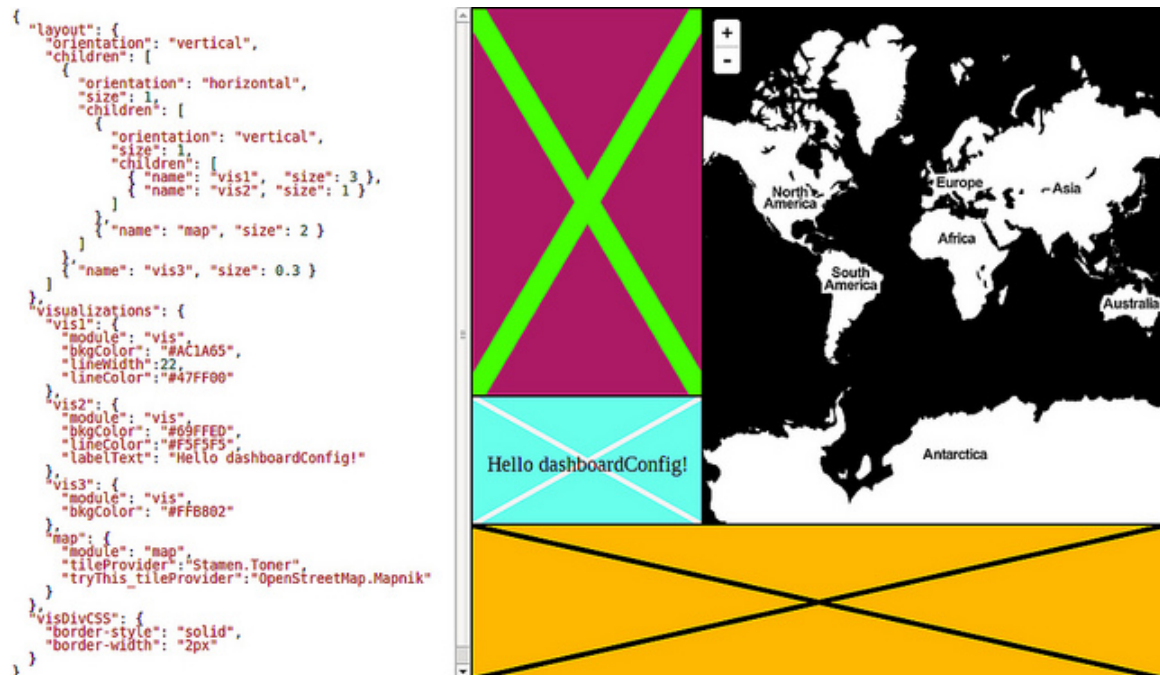


Figure 3.5. The dashboard scaffold prototype, showing an interactive text-based configuration editor on the left, and a sample dashboard on the right. This shows how Leaflet-based maps can be integrated with D3 visualizations using the proposed configuration framework.

does an extraordinary amount of unnecessary work. A more efficient solution would employ a strategy in which only the changes in configuration are propagated through the runtime, while unchanged configuration parameters need not have any effect on the runtime.

All application state configuration updates can be expressed as a series of the following operations:

- `create(alias, module)`
- `destroy(alias)`
- `set(alias, property, value)`
- `unset(alias, property)`

In the context of a user manually updating the configuration at runtime, the difference between two configurations can be computed. The difference results in a structure that can be applied to the runtime essentially as a patch.

3.5 Real-time Synchronization

While Operational Transform algorithms address the general case of many-way synchronization for arbitrary text [35], if we have a simpler application state configuration structure, it is possible to implement simpler many-way synchronization algorithms. Our simple application state configuration difference structure provides an important primitive for real time configuration. The configuration difference objects computed locally can be broadcast to other clients and applied to remote runtimes to achieve real time many-way synchronization.

To implement real time synchronization, consider the application state and its changes as a directed graph $G = (V, E)$. Vertices in the graph represent application states, and each directed edge between nodes u and v represents a change in state. In

a distributed environment with multiple clients, any client can originate a new state. The client that creates a new state provides the associated configuration difference object that can move any client runtime from state u to state v when applied.

Situations may arise in which the source state u of a new state transition does not match the current runtime state. This occurs when two or more clients simultaneously originate conflicting state changes. The distributed synchronization algorithm must guarantee that no matter which order the conflicting state transitions arrive at the server, all clients must ultimately end up in the same state. This requirement can be implemented with simple algorithms that throw away some changes, or more complex algorithms that preserve every change.

Simple distributed synchronization algorithms can be developed that implement either “first one wins” or “last one wins” semantics. First one wins semantics means that the first of several conflicting state transitions that arrives at the server is accepted as the authoritative one and forced upon clients, causing the clients to roll back local conflicting changes. Last one wins semantics means that the last of several conflicting state transitions that arrives at the server is accepted as the authoritative one, causing the server to compute rollbacks and broadcast them to clients. Either way, clients experience loss of changes. In use cases with many concurrent users making changes simultaneously, these semantics would lead to frustrating user experiences, as users must perform their actions twice or more before they are accepted into the authoritative state.

More complex algorithms that involve merging conflicting state transitions can address multiple concurrent changes. These algorithms must merge multiple configuration difference objects and generate a new state originating at the server. The server, with awareness of the states of each client, must send different transitions to each client such that each client ultimately is in the new state that originated from the server.

3.6 History Model

Using the state machine model discussed, transitions can be annotated with an undo operation to enable history graph navigation. Each transition that occurs in the system adds a forward edge (a “do” operation) and a backward edge (an “undo” operation) to the history graph. Numerous algorithms exist for computing the shortest path in an arbitrary directed graph [57]. A shortest path algorithm such as Dijkstra’s can be used to compute the sequence of configuration difference objects to apply to a runtime in order to transition the system from any state to any other state.

3.7 Integration with Web Technologies

To build interactive visualizations for the Web using reactive models, there must be a clearly defined way to interface between reactive models and Web graphics technologies. The primary Web graphics technologies are Canvas and SVG (Scalable Vector Graphics). The Canvas element provides an additional API called WebGL that supports hardware accelerated graphics. The goal of integrating reactive models with Web graphics technologies is to build reusable interactive graphics modules. The modules can be instantiated and linked with specific data to create interactive visualizations.

Instantiations of reusable interactive graphics components in general function in conjunction with a DOM element. Therefore, one way to accommodate any Web technology is to design an API in which a module constructor function creates a DOM element that contains its visual representation. Cascading Style Sheets (CSS) can then be used to position the DOM element. This approach effectively unifies Canvas, WebGL, and SVG, and allows interactive graphics modules authored using different APIs to function together on a single page.

CHAPTER 4

THE UNIVERSAL DATA CUBE

Consider the question “Is there a correlation between the population of a country and its Gross Domestic Product (GDP)?”. The data for population and GDP can only be obtained independently from different sources. Suppose each data set is a relation. To apply visualization or analysis techniques to the data, the two relations must first be joined such that each row in the resulting relation represents a country and contains values for both population and GDP.

Integrating data from a variety of independent sources makes the data more useful, informative, and valuable for visualization and analysis tasks. Existing data integration and multidimensional analysis approaches typically target relational data sources. The data cube concept from the area of Online Analytical Processing (OLAP) is suitable for modeling statistical data sets. These approaches do not readily support integration of pre-computed data cubes, which is applicable for primarily scientific and social data analysis and visualization.

There is immense potential value in data that is not being realized. While publicly available data sets for just about any topic are published on the Web, it is difficult to realize their full value in practice because they exist in many different formats and vocabularies. The heterogeneity of these formats and vocabularies also makes it difficult to combine and analyze these data sets, and hinders the development of general purpose visualization tools.

This chapter presents novel data structures and algorithms for data set representation, integration and querying based on the data cube concept. The proposed

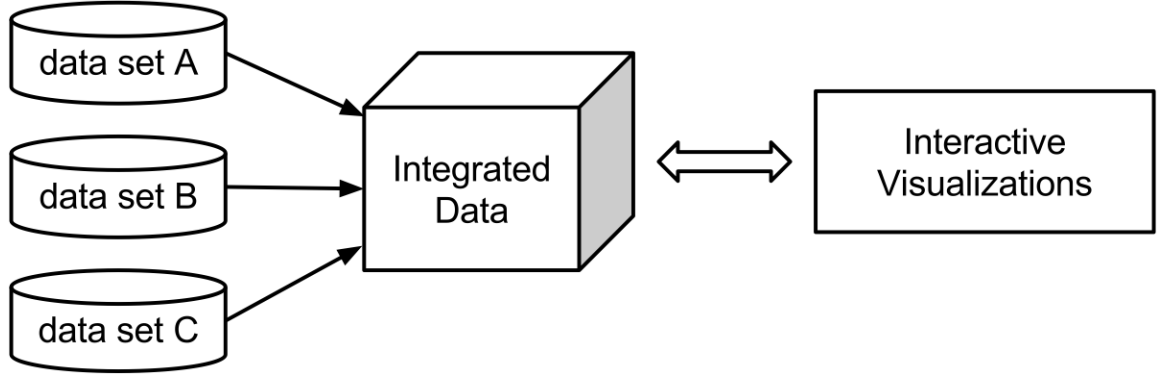


Figure 4.1. The overall context of the UDC as a technology for data integration for the purpose of interactive visualization.

framework is called the Universal Data Cube (UDC). Using this approach, many data sources can be integrated together into a single data structure. General purpose data visualization and analysis tools can then operate by querying the integrated data structure. Figure 4.1 shows the overall flow enabled by the UDC, from disparate data sets to general purpose interactive visualizations.

4.1 Related Work

Existing approaches to representing, visualizing, and analyzing are vast. The relational model is the foundation of relational databases. Data cubes are multidimensional aggregated relations suitable for visualization and analysis. The field of knowledge management is concerned with general data models representing vast collections of concepts and facts. Data integration is an area focusing on techniques for combining data from many sources.

Relational database systems provide a mature data management solution and are widely adopted [120]. Relational algebra is the theoretical underpinning of relational databases [30]. Perhaps the most familiar data representation system today is the spreadsheet, which is capable of representing relations as well as complex operations across data values [50]. Many organizations use spreadsheets, employing Microsoft

Excel, Google Docs, or other tools to manage data or make data available. For example, the United Nations Department of Economic and Social Affairs provides their population statistics in Excel format (see figure 4.8).

The term “data cube” was originally introduced as a relational operator generalizing group-by, cross-tab, and sub-totals [61]. The data cube operator produces relations containing aggregated values from other relations. Data warehouse systems are typically built on the relational model and are augmented by data cubes, also known as OLAP (OnLine Analytical Processing) cubes, for reporting and analysis [31]. The term OLAP stands in contrast to OLTP (OnLine Transaction Processing), which is the part of the data warehouse system that ingests and stores data at the level of individual transactions or events. After the ETL (Extract, Transform and Load) phase of the data warehouse flow, the data is analyzed by computing a data cube from the transactional data.

Data cubes contain summaries of the collection of facts stored in a relational database [27]. For example, a data cube may contain how much profit was made from month to month subdivided by product category, while the relational database may contain the information associated with each individual transaction. Because data cubes provide a higher level of abstraction, they are a widely used method of data abstraction for supporting visualization and analysis tasks. Kimball pioneered the area of “Dimensional Modeling,” which concerns constructing data warehouse schemas amenable to OLAP-based analysis [89]. Data cubes have been implemented in a variety of different systems, so effort has been made to discover unified conceptual or mathematical models that can characterize many implementations [43, 146, 145, 97, 4, 63, 18].

The data cube concept and structure can be used to model existing data as well. Publicly available data sets (often termed “statistical data”) may be considered as pre-computed data cubes if they contain aggregated measures (also called “indicators,”

“metrics,” or “statistics”) across time, geographic space, and other dimensions such as gender, age range, ethnicity, religion, or industry sector.

Existing OLAP technologies assume that the data cubes will be computed from a relational source. They are not designed to handle integration of pre-computed data cubes that may use inconsistent identifiers for common dimensions and measures between multiple data sets. Therefore, the application of the data cube concept to integration and visualization of many pre-computed data cubes, while theoretically plausible, requires the development of novel data structures and algorithms that extend the data cube model to handle integration of pre-computed data cubes that may use inconsistent identifiers for common dimensions and measures. With this approach, it is possible to model many data sets using their shared dimensions and measures. This enables integration of many data sets into a single structure suitable for visualization and analysis.

The Semantic Web is a vision of a “Web of Data” coexisting with the World Wide Web [11]. The basis of the Semantic Web is the Resource Description Framework (RDF) data model, which represents a graph of data in the form of (*subject, predicate, object*) triples. The Semantic Web vision has evolved into the concept of Linked Data, which refers to data that is available as RDF and made available according to common conventions [16, 14]. Any data that can be represented using a relational database can also be represented using RDF [15]. The SPARQL query language for RDF can be used to query and integrate data from multiple sources [118]. Lopez et al. developed an information management system for integrating and analyzing heterogeneous information sources characterizing urban areas [102]. The Semantic Web technology stack contains a method for declaring when different identifiers refer to the same entity and processing queries appropriately to integrate data [67, 45]. While the Semantic Web provides a compelling vision, its adoption is not as widespread as one might expect [103].

The RDF Data Cube Vocabulary is capable of representing data cubes using Semantic Web technologies [40]. The intention of the RDF Data Cube Vocabulary is to provide a common representation and interchange format for statistical data. The RDF Data Cube Vocabulary draws from a previous effort called the Statistical Data and Metadata eXchange (SDMX) initiative that was launched in 2001 by seven organizations working on statistics at the international level [39]. The primary challenges faced when using the RDF Data Cube Vocabulary include transforming to and from well known formats and data models. Salas et al. discussed how data can be transformed from existing OLAP systems or flat files into RDF using the Data Cube Vocabulary and introduced a faceted visualization tool for RDF data cubes [125]. Kämpgen et al. investigated how data represented using the RDF Data Cube Vocabulary can be transformed for analysis using traditional OLAP systems [84]. Maali et al. proposed a pipeline for converting government data into high quality Linked Data utilizing the Data Cube Vocabulary [104].

The field of data integration offers many techniques for combining data from multiple sources based on the relational model [48] as well as from a theoretical perspective [95, 66, 159]. Schema matching is the area of data integration that concerns semantic matching between the attributes of data tables from different sources [119, 53].

Schema matching may be performed manually, but it must be automated to scale to hundreds or thousands of different sources. Numerous approaches for automated schema matching have been proposed [131, 46, 86, 110, 106, 47]. Schema matching approaches aimed specifically at Web- and Ontology-based data integration have also been proposed [71, 113, 49, 107, 83, 114, 144, 148, 115, 82].

Data matching (also known as record linkage) is the area of data integration focusing on resolving different identifiers to the same real-world entity [156, 157, 91, 5, 62]. Record linkage has been applied extensively to public data [81, 80, 75]. Several tools have been introduced that aid users in data integration tasks via a graphical

user interface [29, 85, 51, 60]. Techniques from both of these areas must be applied to integrate data sets from multiple sources and use the proposed unified data model.

4.2 Case Study: Causes of Death

The Centers for Disease Control provides data on causes of death in the US over time. This data set was targeted for visualization as a case study in visualizing public data sets containing pre-aggregated data cubes. A natural way to visualize this data is as a stacked area chart, shown in figure 4.2. The table contained a hierarchy of diseases, and all but the top-level disease categories were removed manually. Selecting the subtree of causes of death to include in the visualization is one example of a task that would be automated with our data representation framework. Next, a JavaScript program was written that pivoted the table from a format where each column was a year to a format where each row is a year, making the table usable by D3.js. This table contained an entry for “all causes,” which was removed manually because it was not appropriate to include visualize.

The mortality data set was published in GitHub Pages using JavaScript Object Notation (JSON) [37] compatible with the structure accepted by D3 [19]. AMD (Asynchronous Module Definition) is a JavaScript pattern for publishing and consuming reusable modules across domains [116]. The mortality data set was published as an AMD module containing JSON data rather than as a CSV or JSON file in order to circumvent the same-origin policy. This allows any Web page to consume the data set, not only pages within the same domain. This method of publishing was chosen because it is a simple way to publish data publicly with zero cost (as GitHub Pages is a free service for Open Source code), longevity, as GitHub is less likely to go offline in the future than a private server, and cross-domain availability (any page can load the module using an AMD loader such as `require.js`). This method of publishing data is also developer-friendly, as most modern developers are familiar with GitHub.

The mortality stacked area visualization highlights several issues that must be addressed when using our proposed data representation framework. The causes of death extracted from the raw data are sometimes too long to use in the visualization as labels due to limited screen real estate. For example, “Symptoms, signs, and abnormal clinical and laboratory findings, not elsewhere classified” is too long to be placed next to its corresponding color in the legend of the visualization, and could be simplified to “Unclassified conditions.” In a data cube model, causes of death would be members in a dimension hierarchy. The labeling issue encountered in the mortality visualization indicates a need to support renaming of members for use as textual elements within visualizations. Since each label refers to a dimension member which may also be a generally well-known concept, the labels on the visualization could, for example, be links to the Wikipedia pages about the various causes of death, such as Cardiovascular Disease. In addition, there are 24 causes of death presented in this visualization using different colors, while the color scales provided by D3 and the color scale library ColorBrewer only support up to 20 colors. This issue indicates that it may be useful to be able to aggregate dimension members automatically as a new “Other” category in certain cases, or allow users to manually select only a subtree of a dimension hierarchy for visualization.

Figure 4.3 shows a sample of the raw data from which the hierarchy of causes of death must be gleaned. In this data, the hierarchy is encoded as an indented tree. Two different characters are used as indentation characters, ASCII codes 32 (space) and 65533 (unknown character). The level of indentation does not use a consistent number of indentation characters per indentation level. For example, the indentation level jumps from 0 to 4 to 7 to 10 to 13. We implemented an algorithm that parses the tree structure from an indented list and outputs the tree in the JSON structure compatible with D3.js hierarchical layouts. The JSON that the D3 Tree Layout algorithm expects is a tree data structure in which each node has a name and

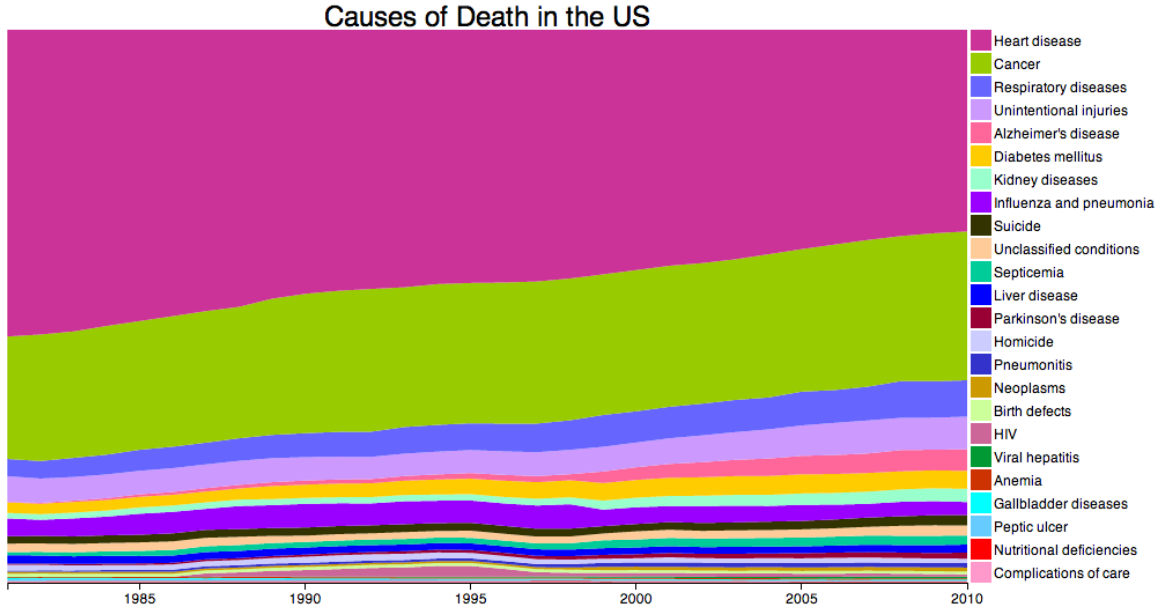


Figure 4.2. A second pass at a stacked area visualization of Mortality data from the US Centers for Disease Control. This version has 25 hand-picked distinguishable colors, a color legend to spread out labels, and shortened labels in some cases.

an array of child nodes. Leaf nodes may have additional quantitative properties to be visualized [22].

Several D3 examples were drawn from to implement the cause of death tree visualization shown in figure 4.4, which uses the ReingoldTilford “tidy” tree layout algorithm [121]. This visualization shows only the hierarchy of causes of death. No numerical values are associated with each node. Notice that the two causes of death that show the highest percentages in our stacked area visualization, Cancer and Cardiovascular Diseases, are the two nodes in the hierarchy that have the two largest subtrees of categorization.

The node-link tree visualization in figure 4.4 demonstrates a visualization technique that can be applied to visualize dimension hierarchies in general. This implementation shows the structure of the hierarchy clearly, but has several drawbacks. Due to the size of the hierarchy, the inclusion of labels for all nodes necessitates small labels that are only legible at high resolution. When a hierarchy scales above certain


```

0 'Major cardiovascular diseases'
4 '    Heart disease'
7 '    Rheumatic fever & rheumatic heart disease'
7 '    Hypertensive heart disease'
7 '    Hypertensive heart and renal disease'
7 '    Ischemic heart disease'
10 '    Heart attack'
10 '    Chronic ischemic heart disease'
13 '    Atherosclerotic cardiovascular disease'
7 '    Heart failure'
4 '    Hypertension'
4 '    Stroke'
4 '    Atherosclerosis'
4 '    Aortic aneurism and dissection'

```

Figure 4.3. A portion of the raw data from the Centers for Disease Control encoding the hierarchy of causes of death. The number of indentation characters is shown on the left, and the content of the “Cause” field from the original CSV file is shown on the right in quotes. Note that there are two different indentation characters used, and the indentation level is not of a consistent multiple. This is one example of an unconventional format that must be parsed into a dimension hierarchy for use within our data representation framework.

thresholds of width and depth, this visualization becomes unwieldy, and labels must be truncated or omitted entirely. This is one example of the scalability issues that must be addressed when developing general-purpose visualization techniques.

The pair of visualizations shown in figure 4.5 is an example of a visualization dashboard with multiple linked views. The tree view shows a single level subtree. Black nodes have children, while white nodes do not. Clicking a black node causes the tree to drill down into the subtree with the clicked node as its root. When this interaction is executed, the stacked area visualization is recomputed to show the new set of disease causes that corresponds to the children of the newly selected node. In this way, the tree visualization provides interactions for drill down and roll up that define the slice of data shown in the stacked area chart.

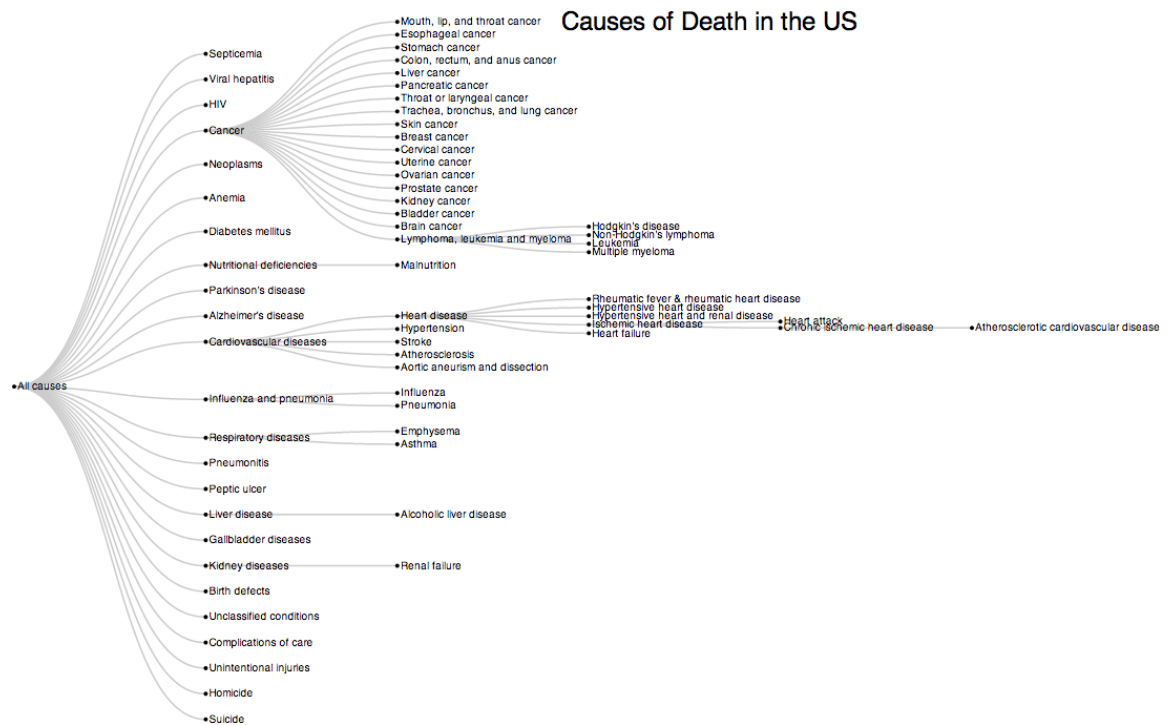


Figure 4.4. A tree visualization of data from the Centers for Disease Control showing the hierarchy of causes of death. This is one example of a visualization that shows the structure of a dimension hierarchy.

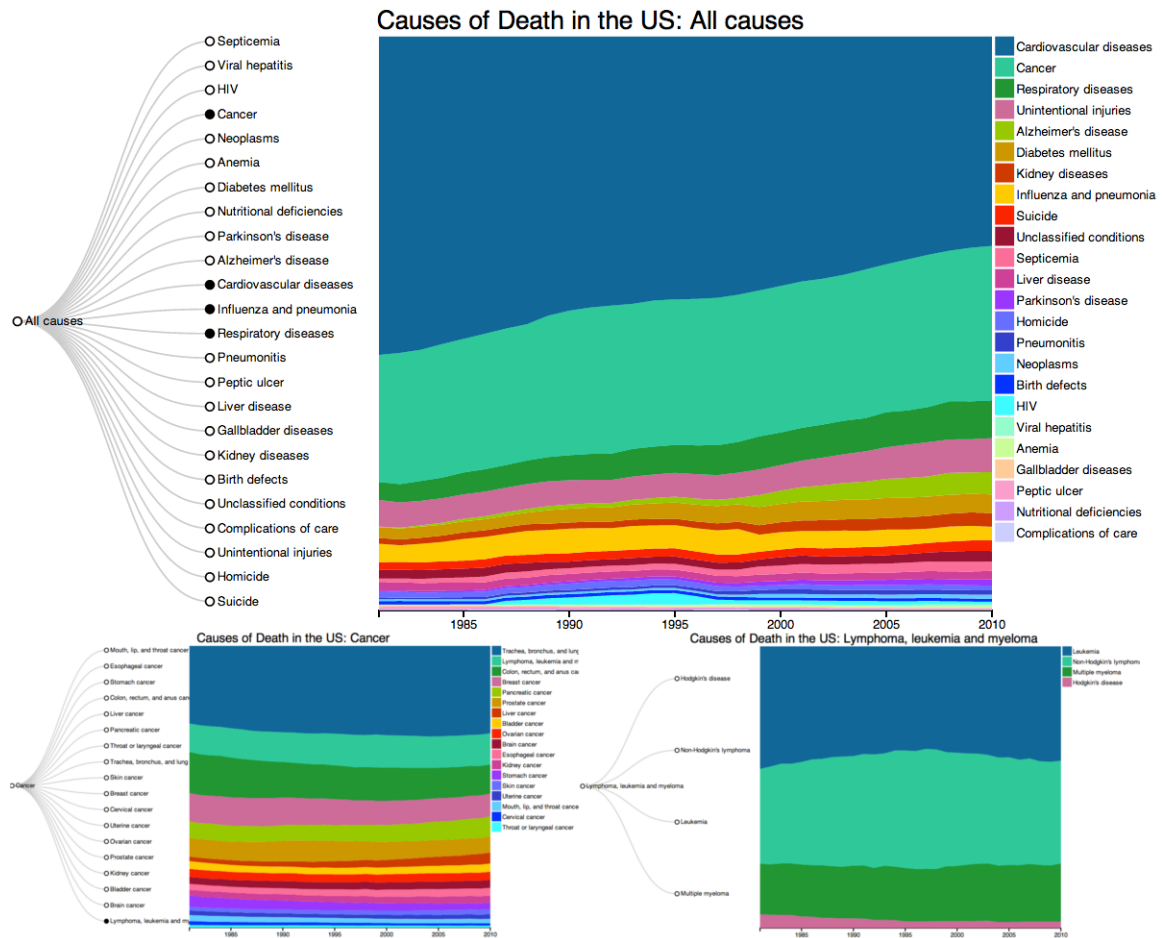


Figure 4.5. Cause of death visualization with two linked views. Navigating up and down the hierarchy by clicking nodes changes the slice of data shown in the stacked area visualization. The top view shows the top-level causes of death. Clicking the “Cancer” node yields the view on the bottom left, which shows types of cancer in the stacked area visualization. Further drilling down to “Lymphoma, leukemia and myeloma” yields the view on the bottom right.

4.3 Data Set Representation

In the sections that follow, capitalization denotes terms that have a concrete and well defined meaning within the UDC data structure. These terms include Data Set, Dimension, Member, Measure, Dimension Column, Measure Column, Cube Data Set, Concordance Data Set, Codelist, and Code. Each of these terms will be introduced one by one in terms of what role they play in the UDC data structure. Italicized terms are functions that perform the algorithms associated with data integration and querying. These algorithms operate in terms of the UDC data structure.

A Data Set is a relation with associated metadata that provides context for its interpretation. Tables 4.2, 4.1 and 4.3 are concrete examples of Data Sets, showing both their relations and metadata. The relation contains the core table of the Data Set. The metadata specifies the interpretation of the relation in terms of Dimensions and Measures. Columns in relations can be annotated in the metadata as either Dimension Columns or Measure Columns. A Dimension Column contains Codes from a single Codelist that refer to Members of a given Dimension. A Measure Column contains numbers representing values for a given Measure. A scaling factor can be associated with each Measure Column to account for varying scales for the same Measure across different Data Sets.

To integrate Data Sets together, we need two kinds of Data Sets, Cube Data Sets and Concordance Data Sets.

A Cube Data Set contains a relation that represents a data cube using a star schema. Certain columns represent categorical Dimensions, while others represent quantitative Measures. Dimensions are sets of distinct entities that may be unordered, ordered, or hierarchical. Each distinct entity of a Dimension is called a Member. A row in the relation of a Cube Data Set represents a unique set of Members, one from each Dimension of the Cube Data Set, in its Dimension Columns. The unique set of Members represented in each row is called a Cell, which acts like an address in

Table 4.1. A Cube Data Set about Population.

table	countryCode	year	pop		
	356	1960	449595.489		
	356	2010	1205624.648		
	156	1960	650680.114		
	156	2010	1359821.465		
	840	1960	186361.893		
	840	2010	312247.116		
dimensions	column	dimension	codelist		
	countryCode	Space	UN M.49		
	year	Time	Year		
measures	column	measure	scale		
	pop	Population	1,000		

Table 4.2. A Cube Data Set about GDP.

table	countryCode	year	gdp		
	IND	1960	37679274491.2745		
	IND	2010	1708450861364.17		
	CHN	1960	61377930682.0013		
	CHN	2010	5930529470799.17		
	USA	1960	520531181568		
	USA	2010	14958300000000		
dimensions	column	dimension	codelist		
	countryCode	Space	ISO3		
	year	Time	Year		
measures	column	measure	scale		
	gdp	Gross Domestic Product	1		

Table 4.3. A Concordance Data Set linking equivalent terms used by different cubes referring to countries.

table	countryName	unCountryCode	alphaCode		
	India	356	IND		
	China	156	CHN		
	United States	840	USA		
dimensions	column	dimension	codelist		
	countryName	Space	UN Geoname		
	unCountryCode	Space	UN M.49		
	alphaCode	Space	ISO3		

the data cube. Measures are aggregated quantitative properties that can be assigned to Cells in the Measure Columns of the relation. A Cube Data Set contains a set of Observations that draw from a common set of Dimensions and Measures. An Observation links a Cell with concrete values for each Measure. Each complete row of the relation represents a single Observation. Many Cubes can reference the same Dimensions, Members, Cells, and Measures, whereas each Observation belongs to exactly one Cube. Table 4.4 provides examples for each of the concepts introduced.

Table 4.4. Examples for UDC Concepts.

Concept	Example(s)	Description
Dimension	Space, Time	A hierarchy of entities, identified by a string name.
Measure	Population, Gross Domestic Product	A kind of numeric value that can be assigned to Cells by Observations.
Cube	The population of India in 1970 was 555.2 million The population of India in 2010 was 1.206 billion The population of China in 1970 was 818.3 million The population of China in 2010 was 1.338 billion	A set of Observations that draw from a common set of Dimensions and Measures.
Observation	The population of India in 1970 was 555.2 million.	An object that represents a mapping from a Cell to a numeric value for a specific Measure.
Cell	India in 1970, China in 2010	A unique set of Members.
Member	India, China, 1970, 2010	A single entity within a Dimension hierarchy, identified by a Code.
Codelist	Country Name, Country Code	A controlled vocabulary for identifying Members.
Code	India, in	A single string identifier within a Codelist that identifies a particular Member.
Concordance	Country Name Country Code India in China cn	A relation that defines equivalences between Codes.

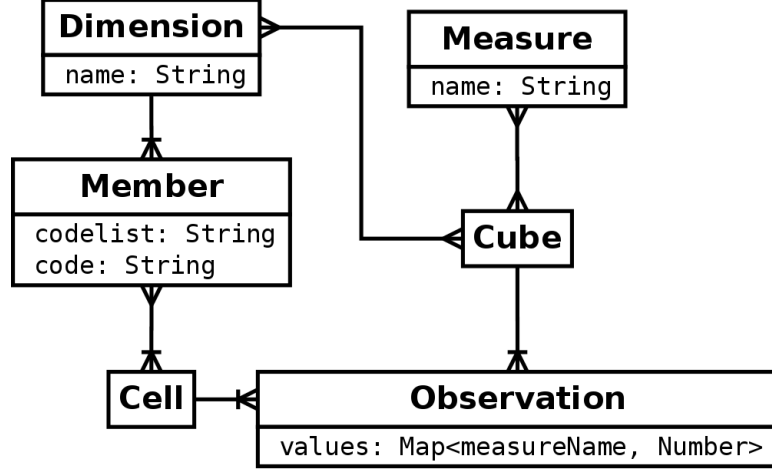


Figure 4.6. The Logical Data Structure expressing the representation of Cube Data Sets within the UDC data structure, using the visual data modeling language described in [26].

A Member is a unique entity within a Dimension, identified by a single Code. Codelists are controlled vocabularies for referring to Members. A Code is a string within a Codelist that refers to a specific Member. Each Codelist contains Codes that refer to Members of a single Dimension. The metadata of a Cube Data Set defines how some columns of the relation can be interpreted as Dimensions. This metadata is used for transforming the strings in each row of Dimension Columns into Member instances within the UDC data structure.

The Logical Data Structure (LDS) shown in figure 4.6 expresses the essential concepts of the UDC data structure and their relationships. This data structure is used to represent Cube Data Sets after they are loaded into memory. Cube Data Sets loaded into this data structure can be integrated together and queried for interactive visualization.

The following functions transform Data Sets into in-memory representations and help integrate multiple Cube Data Sets together.

- *createCube(dataset) → cube*

- *createThesaurus(datasets) → thesaurus*

The *createCube* algorithm creates a Cube (from figure 4.6) from a Cube Data Set. This procedure must iterate through each Dimension Column and Measure Column of the Data Set to compute the Dimensions and Measures associated with the Cube. Once the metadata has been processed, each row in the relation of the Data Set is transformed into a single Observation. In computing Observations, the requisite Members and Cells are also defined within the in-memory data structure.

The *createThesaurus* function generates an index that can be used to implement the *canonicalizeMember* function necessary for merging Cubes. The set of Data Sets input to the *createThesaurus* function should each be a Concordance Data Set. This means they each have only Dimension Columns that refer to Members of the same Dimension using different Codelists. The relations of Concordance Data Sets serve only to define equivalences between Codes from different Codelists that refer to the same Member. These relations are transformed into an index that maps each $(code, codelist)$ pair to a set of other $(code, codelist)$ pairs that refer to the same Member. The index of equivalence classes can be used in conjunction with a canonical Codelist for each Dimension to implement the *canonicalizeMember* function. A canonical Codelist can be chosen by sorting all Codelists used in a given Dimension alphabetically and choosing the first one.

4.4 Querying Data Sets

Cubes can be queried by interactive visualizations. Interactive data visualizations need to make a number of different queries for generating user interface components, labels, axes, scales, and visual marks. We assume that a visualization has access to a single Cube containing the integrated data. Starting from the Cube object the following functions provide all queries necessary for general purpose interactive visualizations to operate. This syntax represents functions in terms of their name

(before parentheses), what they accept as input (in parentheses), and what they yield as output (after the arrow). Types are denoted as lower case counterparts to UDC concepts, because they represent concrete instances of those concepts.

- $listDimensions(cube) \rightarrow [dimension]$
- $listMeasures(cube) \rightarrow [measure]$
- $listObservations(cube) \rightarrow [observation]$
- $getValue(observation, measure) \rightarrow \mathbb{R}$
- $getCell(observation) \rightarrow cell$
- $member(cell, dimension) \rightarrow member$
- $slice(cube, cell) \rightarrow cube$

The *listDimensions*, *listMeasures* and *listObservations* functions list which Dimensions, Measures, and Observations, respectively, are associated with the given Cube. The *getValue* procedure extracts the numeric value for a given Measure from a given Observation, by evaluating its *values* property (which maps Measures to numeric values). The *getCell* procedure reads from the data structure which Cell is associated with the given Observation. The *member* procedure extracts from the data structure which Member is contained within the given Cell for the given Dimension. The *slice* procedure implements the traditional OLAP slice operation defined in [43].

4.5 Integrating Data Sets

The following procedures relate to integrating Cubes:

- $canonicalizeMember(thesaurus, member) \rightarrow member$
- $canonicalizeCube(thesaurus, cube) \rightarrow cube$

- *mergeCubes(cube, cube) → cube*
- *integrate(datasets) → cube*

The *canonicalizeCube* function transforms a Cube such that all of its Observations use canonicalized Members. When two Data Sets use different Codes to refer to the same Members, the Cubes generated directly from the Data Sets define their Observation domains (Cells) in terms of different Members. A Thesaurus can be used to implement the *canonicalizeMember* procedure, which resolves equivalent Members referred to using Codes from different Codelists. The *canonicalizeCube* function invokes *canonicalizeMember* as a subroutine to normalize the Observations of the Cube. After running *canonicalizeCube* on two Cubes that use different Codelists, both resulting Cubes use the same Codes to refer to the same Members. This step enables the two Cubes to be merged using *mergeCubes*.

The *mergeCubes* procedure joins the Observations from two Cubes by matching on their Cells. Each Cell is identified by a unique set of Members. Cells from both Cubes match because the *canonicalizeCube* procedure was invoked previously on each input cube. In implementing *mergeCubes*, one could employ an inner join strategy, which would yield a Cube with no missing data but may not include all of the original data, or an outer join strategy, which would yield a Cube with missing data (Observations with missing values for some Measures) but would include all of the original data.

Figure 4.7 shows the *integrate* algorithm for integrating many Data Sets. This algorithm first transforms all Cube Data Sets in the input *datasets* array into Cubes using *createCube*, and constructs a Thesaurus from all Concordance Data Sets in the input *datasets* array using *createThesaurus*. A map-reduce pattern is then applied to integrate all Cubes. The map portion of the algorithm applies the *canonicalizeCube* function to all Cubes, using the already created Thesaurus to canonicalize Members used in each Cube. The reduce portion of the algorithm applies the *mergeCubes* func-

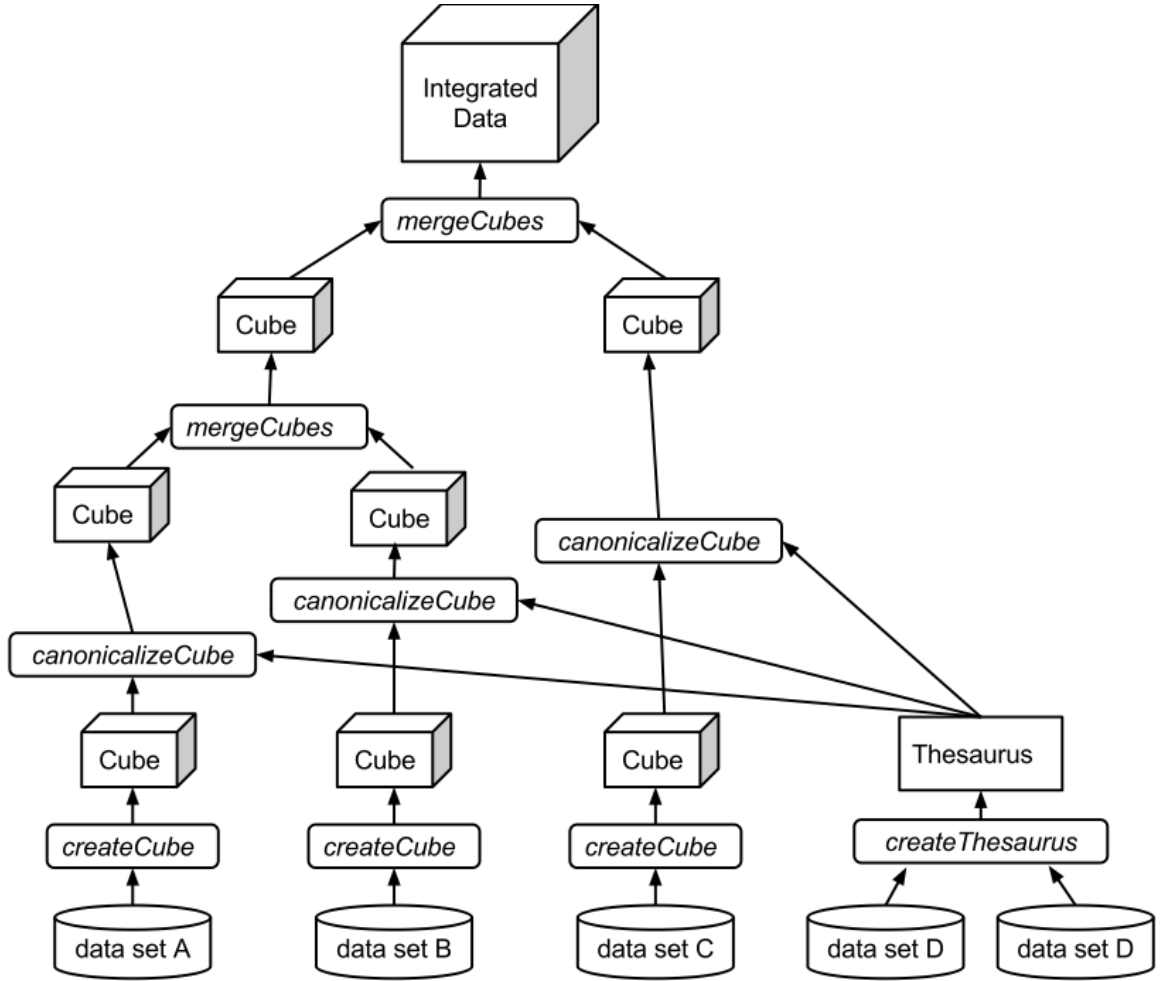


Figure 4.7. The flow of the *integrate* algorithm for integrating data.

tion to merge all canonicalized Cubes recursively. The result from the entire algorithm is a Cube that contains data from all of the input Cubes combined and integrated properly. The resulting Cube is suitable for input into interactive visualizations.

4.6 Limitations of Data Cube Representation

Many, but not all, data sets can be modeled as data cubes. Since data cubes are only capable of representing data that has been aggregated along categorical dimensions, there are many classes of data that do not fit within the conceptual framework of data cubes. For example, a database containing the details of transactions in a

supermarket would not be appropriate to model as a data cube. Each entry of a customer purchase may contain a listing of items purchased, how it was paid for, and the date and time the purchase was made. This kind of data fits well into the relational model, but is not appropriate to model as a data cube. Data cubes represent only aggregated summaries, not individual events. In the case of grocery store database containing the transactions for many grocery stores in different regions, while the individual transaction entries cannot be modeled as a data cube directly, a data cube can be constructed from the transactional data by aggregating measures such as “amount paid” and “number of items purchased” along dimensions such as “time”, “region” and “product category”. This is a typical data warehouse scenario, where a business aggregates transactional data into a data cube in order to analyze company activities in a summary view.

The key characteristics that allow a given data set to be modeled as a data cube are as follows:

- The data set contains numeric fields that represent aggregated summaries using sum, average, or some other aggregation operator (measures).
- The measures of the data set are aggregated along one or more sets of discrete categories or entities (dimensions).

Dimensions can be either unordered, ordered, or hierarchical. Dimensions include Space, defining hierarchies of geospatial regions, Time, defining intervals in time, or of any arbitrary collection of categories. Examples of dimensions other than Time and Space include Gender, Ethnicity, and Industry. The Space dimension can be decomposed using several alternative strategies. The most common spatial decomposition found in data is along geopolitical boundaries. Another way to decompose spatial regions is according to a quadrilateralized spherical cube, also called a quad sphere [142]. The quad sphere approach provides uniformly distributed regions at

multiple levels of detail, which makes it appropriate for presenting aggregated summaries of more detailed data such as billions of points on Earth or satellite imagery data [152]. Yet another alternative geospatial partitioning strategy is by global river basins. Partitioning data by river basins makes sense for climate and weather related data [9].

Data sets that have the following qualities may not be modeled directly as data cubes.

- The data set represents a graph. Graph data such as social network connections or links between Web pages is not supported by the data cube model.
- The data set contains relational data with a one-to-many relation. For example, a database of transactions in a grocery store where one transaction has many items. Items containing lists of other items cannot be represented using the data cube model. However, one may consider transforming data sets like this such that the nested lists are summarized by some measures (such as total cost or number of items) so the data cube model can be applied.
- The data set contains entries for individual discrete events or transactions. Data sets with this quality cannot be modeled directly as data cubes, however it may be possible to compute data cubes by aggregating them using OLAP techniques from data warehousing.

Although data sets with the above qualities may not be modeled directly as data cubes, it may be possible to compute data cubes that summarize data sets like these. For example, the BrightKite data set which was originally structured as a graph [28] can be aggregated using the data cube approach then visually analyzed [99, 100]. Any graph data set in which nodes contain metadata such as location and time can be aggregated along Space, Time and other dimensions to form a data cube.


 United Nations Population Division Department of Economic and Social Affairs									
World Population Prospects: The 2010 Revision									
File 1: Total population (both sexes combined) by major area, region and country, annually for 1950-2100 (thousands)									
Estimates, 1950-2010									
POP/DB/WPP/Rev.2010/02/F01									
April 2011 - Copyright © 2011 by United Nations. All rights reserved									
Suggested citation: United Nations, Department of Economic and Social Affairs, Population Division (2011). <i>World Population Prospects: The 2010 Revision, CD-ROM Edition</i> .									
Major area, region, country or area	Notes	Country code	Total population, both sexes combined, as of 1 July (thousand)						
			1950	1951	1952	1953	1954	1955	
WORLD		900	2 532 229	2 580 960	2 628 448	2 675 766	2 723 726	2 772 882	2 8
More developed regions	a	901	811 187	820 861	830 924	841 203	851 569	861 930	8
Less developed regions	b	902	1 721 042	1 760 099	1 797 524	1 834 563	1 872 158	1 910 951	1 9
Least developed countries	c	941	196 088	200 293	204 457	208 680	213 041	217 594	2
Less developed regions, excluding least developed countries	d	934	1 524 954	1 559 806	1 593 067	1 625 883	1 659 117	1 693 357	1 7
Less developed regions, excluding China		948	1 160 539	1 183 889	1 208 654	1 234 797	1 262 281	1 291 068	1 3
Sub-Saharan Africa	e	947	186 103	189 777	193 634	197 666	201 867	206 235	2
AFRICA		903	229 895	234 594	239 501	244 621	249 960	255 521	2
Eastern Africa		910	64 757	66 196	67 686	69 228	70 826	72 483	
Burundi		108	2 456	2 505	2 551	2 595	2 640	2 687	
Comoros		174	156	160	164	168	172	175	
Djibouti		262	62	63	65	67	68	70	
Eritrea		232	1 141	1 162	1 185	1 210	1 236	1 264	
Ethiopia		231	18 434	18 788	19 151	19 522	19 894	20 268	

Figure 4.8. The United Nations Population Prospects data set [112], made available in Excel format. This is an example data set that can be imported into our data structure and integrated with other data sets.

4.7 Proof of Concept

The United Nations (UN) Population Prospects Data Set provides population data for each country of the World over time. A small sample of this data set is shown in table 4.1. This data set was downloaded as an Excel spreadsheet (shown in figure 4.8), exported as a Comma Separated Value (CSV) file, then transformed to be in the format expected by a data cube dataset (one row for each Observation). This data set uses the UN M.49 country codes to refer to countries. Figure 4.9 shows a slice of this data set showing population values over time for the entire world. This visualization gives a broad overview of the data. By drilling down, lines in the line chart can be made to represent individual countries.

The World Bank publishes publicly available data about a number of topics. One dataset from the World Bank contains the Gross Domestic Product (GDP) of each

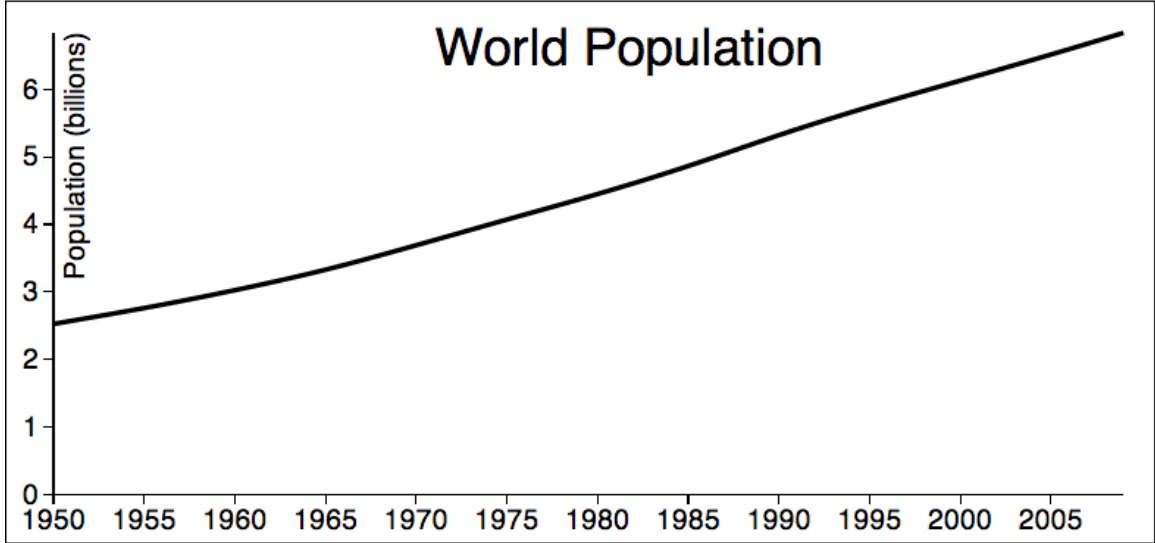


Figure 4.9. A timeline visualization of the United Nations Population Estimates data set. This shows the population of the entire world from 1950 to 2010.

country over many years, shown in figure 4.10. A small sample of this data set is shown in table 4.2. This data set uses the ISO3 standard Codelist for countries.

We applied the UDC approach to modeling and integrating the UN Population Prospects data set and the World Bank GDP data set. The resulting integrated Cube was visualized as a scatter plot, shown in figure 4.11. In this visualization, Population is mapped to the X axis, GDP is mapped to the Y axis, and each dot represents a country. Both axes are log normalized in order to spread the data. This plot shows that there is a correlation between Population and GDP, and that both measures follow roughly a Power Law distribution. This demonstrates a proof of concept implementation of the UDC data structure and algorithms for integrating and visualizing public data.

Consider the steps required to produce the scatter plot visualization of the integrated Population and GDP Cube shown in figure 4.11. First, the dimensions of the integrated Cube, namely Time (Years) and Space (Countries), can be computed using the *listDimensions* function. Since we want each dot to represent a country,

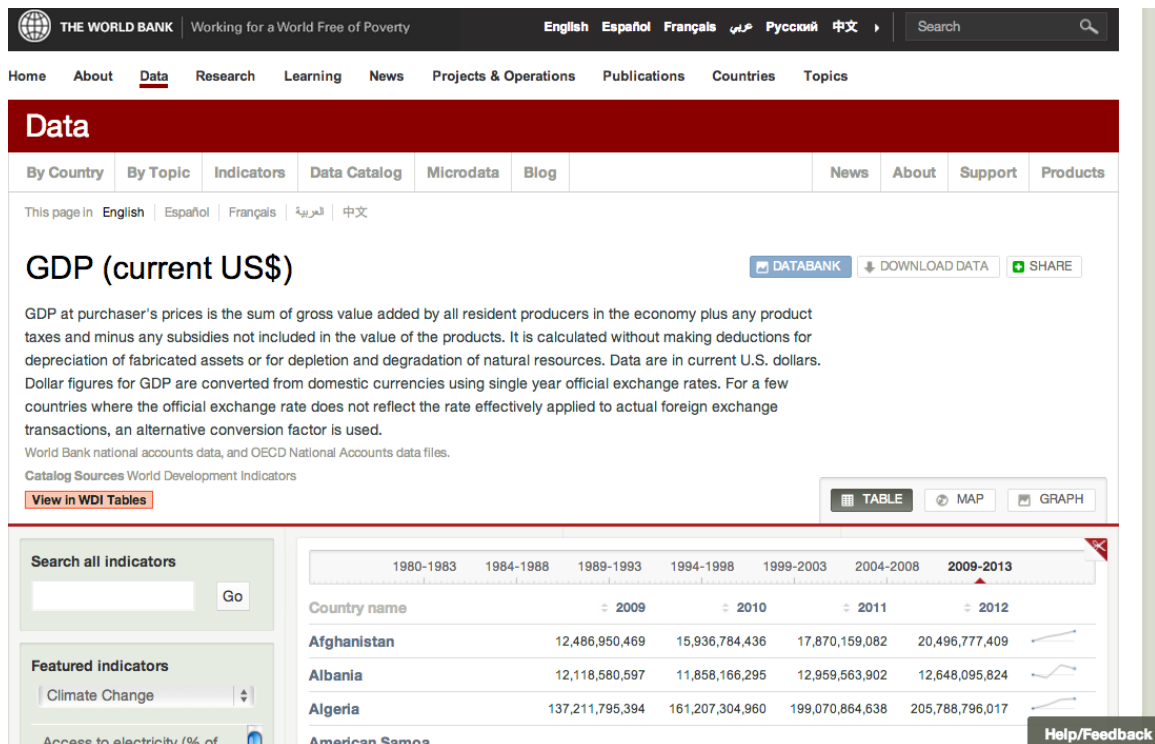


Figure 4.10. The World Bank GDP Data Set.

the Cube must then be sliced using the *slice* function such that it contains only data for a single year. Next, the X and Y scales must be defined. The *listMeasures* procedure can be used to generate a list from which users can choose which measures to assign to the X and Y axes. Once X and Y measures are chosen, the *listObservations* procedure can generate the list of Observations to transform into visual marks (one for each Country). To compute the domains for the X and Y scales, the Observations can be queried for their X and Y measure values using the *getValue* procedure. The minimum and maximum values returned for the X and Y measures can define the domains for the X and Y scales. Once the scales have been defined, the Observations can be iterated over once more to generate the complete visualization. This is one example of how the given query algorithms for Cube Data Sets can be used for visualization.

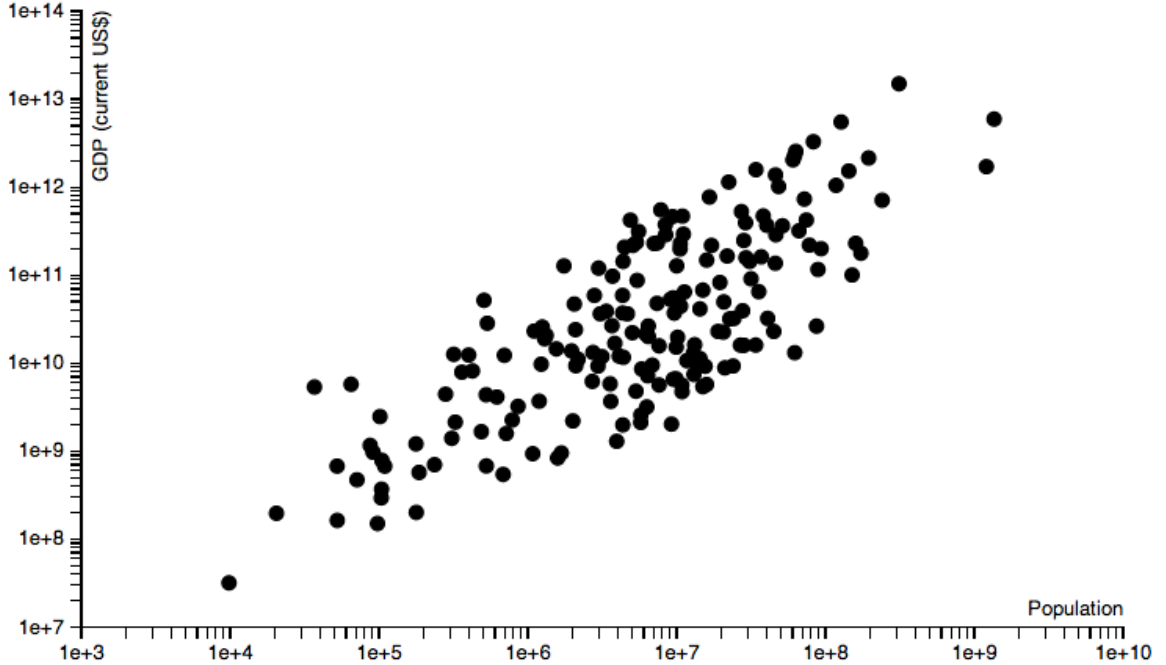


Figure 4.11. A scatter plot visualizing two data sets integrated together. The X axis shows Population, drawn from a United Nations data set, and the Y axis shows GDP, drawn from a World Bank data set. Each dot represents a country.

The *slice* procedure can be utilized for developing visualizations with multiple linked views. This is when one visualization shows an overview, and interactions in that visualization can define how the input data for another visualization is sliced before it is visualized. We have implemented a prototype of this concept which uses a linked Choropleth Map and Line Chart, shown in figure 4.12. Zooming in the Choropleth Map defines the set of countries represented as lines in the Line Chart. Selecting a year in the Line Chart causes the Choropleth Map to show data for that year only.

4.8 Crowdsourcing Data Experiment

Rather than manually curating data, a crowdsourcing approach can be taken to data collection for the UDC. We have performed an initial experiment to test the feasibility of this approach. Amazon Mechanical Turk supports assignment of tasks,

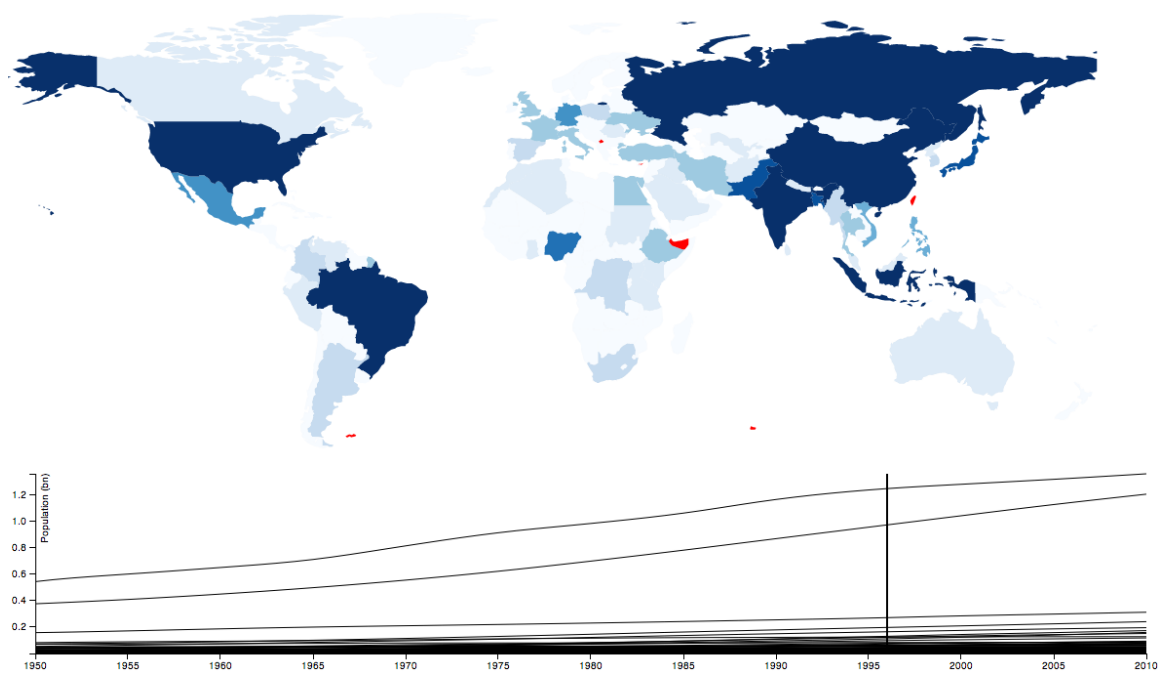


Figure 4.12. A linked line chart and Choropleth map showing population data from the United Nations. Zooming in the map filters the line chart. Selecting a year in the line chart causes the map to show data from that year. This demonstrates how the *slice* procedure can be utilized for generating interactive visualizations with linked views. Red indicates missing data.

called “Human Intelligence Tasks” or HITs, to workers who get paid small amounts (on the order of cents) to execute the tasks. To populate the UDC using Mechanical Turk, HITs can be devised that ask workers to find an answer to a simple question like “What was the population of India in 1950?”. This question is an instance of a more general form “What was the $\{\text{measure}\}$ of $\{\text{place}\}$ in $\{\text{time}\}$ ”. By enumerating possible values for $\{\text{measure}\}$, $\{\text{place}\}$, and $\{\text{time}\}$, responses to such HITs can populate large regions of the UDC.

To test the crowdsourcing data collection approach, an experiment was performed using Amazon Mechanical Turk. In this experiment, $\{\text{measure}\} = \text{population}$, $\{\text{place}\} = \{\text{India, China, United States}\}$, and $\{\text{time}\} = \{1950, 2010\}$. The results contained between 7 and 10 responses from multiple workers for each combination of place and time. By taking the mode (most frequently occurring value) of the worker submissions for each combination of place and time, the following table was generated:

Place	Time	Population	Source URL
India	1950	369880000	www.geohive.com/earth/population3.aspx
China	1950	563000000	geography.about.com/od/populationgeography/a/chinapopulation.htm
USA	1950	150697361	en.wikipedia.org/wiki/1950_United_States_Census
India	2010	1150000000	www.indiaonlinepages.com/population/india-population.html
China	2010	1339724852	en.wikipedia.org/wiki/Demographics_of_China
USA	2010	308745538	en.wikipedia.org/wiki/United_States_Census

Figure 4.13. Initial results from an experiment in crowdsourcing public data using Amazon Mechanical Turk.

In the table shown in figure 4.13, each row represents an observation within the data cube. The values in the Place column refer to members of the Space dimension. The values in the Time column refer to members of the Time dimension. The values in the Population column assign numeric values to cells (combinations of Space and Time members) for the Population measure. This initial result demonstrates the feasibility of crowdsourced data collection for the UDC.

4.9 Summary

This chapter introduced novel data structures and algorithms for integration of multiple data sets, and for querying the merged data structure for use in interactive visualizations. We call this collection of data structures and algorithms the Universal Data Cube (UDC). This approach functions by using concordances to canonicalize identifiers used across data sets, then merging data cubes in a map-reduce fashion. The resulting integrated structure can be queried for generation of interactive visualizations. Future work will focus on developing reusable interactive visualizations that can use the UDC data structure as input, and on a framework for easily assembling linked views based on the UDC data structure.

CHAPTER 5

CONCLUSION

Data today is growing in both size and variety. By modeling the data within a data cube based framework, it is possible to tame the vast collection of various representations of complex data into a predictable structure. This simplified data model imposes structural restrictions on the data, and in doing so makes the data more amenable to interactive visual analysis.

5.1 Contributions

The contributions of this dissertation are novel data structures and algorithms covering a broad range of the data visualization pipeline. The Universal Data Cube data model is an attempt at providing a data model that both captures the essence of many data sets and exposes a simple and predictable data structure upon which interactive visualization systems can be built. The Reactive Visualization approach is the first to synthesize the Model View Controller paradigm with functional reactive programming to construct reusable interactive visualization components. Dynamic visualization configuration allows reusable visualizations to be instantiated within the context of many data sets and allows users to collaborate in real time over interactive visualizations. Taken together, these contributions represent a new approach to the process of integrating and visualizing data.

Reactive models provide the foundation for reactive visualizations. This approach allows the construction of reusable reactive flows that encapsulate elements of interactive visualizations. Using reactive flows, much of the data visualization pipeline

can be implemented. Reactive flows enable representation of update flows from data and configuration through to visualization scales, axes and visual marks. In theory, any imaginable visualization technique can be implemented as a reactive visualization component. The prototypes presented in this dissertation serve as a starting point for a growing catalog of Open Source visualization modules.

Interactions such as brushing, picking and zooming can also be encapsulated using reactive models. The interactions in one visualization can be used to drive interactive changes in other visualizations. In this way visualizations with multiple linked views can be constructed. Interaction schemes for linked views include dynamic filtering, slicing, linked selection and linked probing. For example, zooming in a map can filter the regions used to filter the input to a line chart of population. This technique of using interaction for dynamic slicing can be used to build data cube exploration tools that replace small multiples with interactive linked views.

An application state model based on reactive models provides the foundation for a collaborative visual data exploration platform. Application state configuration based on JSON enables configuration of visualizations with multiple linked views. A unique runtime engine is introduced for configurable applications that dynamically loads required modules. Using this framework it is possible to author and evolve instantiations of reusable visualization components with linked views. The application state model also affords construction and navigation of history graphs supporting undo, redo, and view sharing.

The Universal Data Cube framework proposes a framework for representing data based on the data cube model. This framework is “universal” in that it is capable of representing aggregated summaries of any measurable quantity over limitless time and space. The Time and Space dimensions provide conceptual delineations between regions of hierarchical time and space. Measures provides windows into phenomena occurring within certain regions of time and space. In theory, all measurable quanti-

ties that summarize events and phenomena can be represented using this framework. The limited Open Source proof of concept implementation of the UDC framework is proposed as the seed for a large ongoing project aimed at curating and exposing public data.

By combining the data curated within the UDC and the proposed visualization and collaboration frameworks, the aim is to provide a complete platform for data visualization. The hope is that this platform will enable Web users and content creators to easily investigate any data of interest using interactive data visualization. Application areas for this technology include commercial data analysis, business intelligence, education, journalism and public policy.

5.2 Future Work

The realization of this work so far is far from the complete vision. The vision is to establish a Wikipedia-like platform for collaborative data visualization. The technologies presented in this dissertation provide the foundation for such a system, but more work needs to be done to glue the pieces together into a complete and coherent suite of tools.

The reactive model solution introduced has the potential to be formalized as a time tick based execution system. This kind of formalization would allow a rigorous asymptotic analysis of reactive models based on their data flow graph structure. For example, one could quantify the influence of various graph characteristics such as total number of nodes and breadth first search tree distance from a node that is the source of a change.

The Universal Data Cube data model has the potential to be implemented within a database. This would involve the construction of a database schema that persists the UDC model and allows users to query the data present. The overall effect of maintaining a UDC database is that common dimensions and measures will become

clear. Given a collection of 100 data sets, a UDC database could keep track of the total list of all dimensions and all measures. For each dimension, the database could tell you how many data sets use it, and what the most common key codes are. The database could also provide information about measures such as how many measures there are, how many data sets cover each measure, and what dimensions and members are covered by each measure. These kinds of queries could be coupled with visualization interfaces to present a rich overview of a large collection of data and support unprecedented visual data exploration capabilities.

The envisioned collaboration framework would be a database driven Web application enabling users to import their own data, add their own reusable visualization components, and collaboratively author interactive visualizations. User interfaces supporting these operations must also be developed. For importing data sets, a simple solution similar to the ManyEyes data import user interface would suffice [147]. For adding reusable visualization components, an in-browser code editing tool with versioning would be required (similar to JSBin). For collaboratively authoring interactive visualizations, the dynamic configuration structure can be linked with an operational transformation system such as ShareJS or Firebase. States of interest must be persisted to a database for later viewing or further editing. Existing collaboration platforms that already track data sets such as OpenChorus [93] could be extended to include interactive visualizations. The visualization state created by users can be simply persisted as a string in the application database.

The UDC visualization technology can be extended to accommodate real time updating data. This could be accomplished by polling for data updates or using a server push technology such as WebSockets. For updating data, the data values of the most recent time slice of the data cube may change. For granular time slices such as hours or days, new Time dimension members are created as time passes, so

new slices of the data cube could be sent from the server to clients as they become available.

Mobile technology could be one of the largest targets for interactive visualization technology. For example, many readers of popular news feeds such as the New York Times consume their content using mobile devices such as smart phones, phablets and tablets. The same is true for E-Books and blogs. With this in mind, considerations must be made regarding how to tailor the proposed interactive visualizations to mobile devices with touch screen interfaces and variable resolution.

Only a few reusable visualization components have been implemented and a few data sets imported into the framework so far. The following sections discuss development of more reusable visualization components, more interaction techniques, and importing of more data sets into the UDC framework.

5.3 Visualizations and Interactions

The complete set of visualization techniques suitable for adaptation as reusable components is vast. Bertin enumerates the numerous possibilities for visually encoding data [13]. This is why an expandable user-driven platform is more suitable than a fixed set of visualizations. As a first step toward the goal of an open platform with the potential to accommodate any imaginable visualization technique, an initial set of visualization techniques are presented here in table 5.1.

These visualizations can be augmented by interaction techniques such as brushing, picking, hovering, pan and zoom. Each of these interactions results in a user defined subset of the data visualized. The output from the interaction in any visualization can be used to slice or filter the input of another. Brushing involves the interactive visual definition of a rectangular region that defines intervals in data space. The intervals defined by a brush region can be used to filter the original data, allowing users to interactively define subsets of the data. Picking involves tapping or clicking

on a single visual mark. Hovering (also called probing) selects the single visual mark closest to the mouse. Hovering can use a Voronoi overlay to improve performance. Panning and zooming can be used on a geographic map to define a subset of regions. Conceivably, pan and zoom could also be implemented in other visualizations such as scatter plot.

5.4 Data Sources

There are numerous rich data sources on the Web that are suitable for import into the Universal Data Cube framework for visualization. So far only a few data sets have been actually imported, however many have been evaluated and identified as candidates. Some data sources provide archives of historical data, while others are vastly large and expose the data via a Web API. Crowdsourcing platforms also present immense potential for finding data sources and curating data.

The Pew Global Religious Landscape data set provides data about religious composition by country. This data set covers dimensions of Time (only the 2010 slice is provided), Space (countries), and Religion. The measure can be modeled as Population, as the addition of the Religion dimension effectively is a filter through which to view the population of a place. This data set can be enhanced by integration with a data set providing population data for each country of the World in 2010.

For each country, it is possible to visualize the data cube slice for that country as a bar chart in which each bar represents a religion. Therefore a visualization with linked views can be assembled in which picking interaction in a choropleth map can drive the country used as input for the bar chart. Clicking on each country would show its religious profile as a bar chart. This linked visualization would use the map to visualize and navigate the Space dimension, and the bar chart to visualize the Population measure by the Religion dimension. Together, the map and bar chart could visualize a data cube with 2 Dimensions (Space, Religion) and 1 Measure (Popula-

Table 5.1. Reusable Data Cube Visualizations.

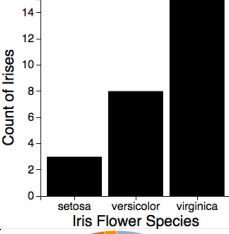
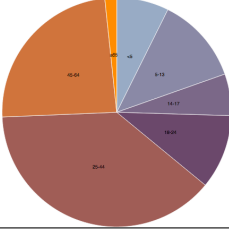
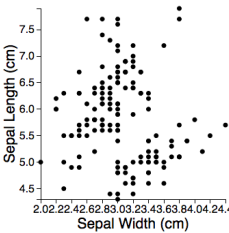
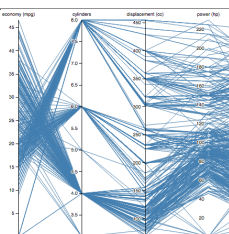
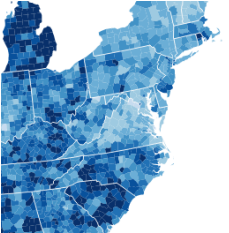
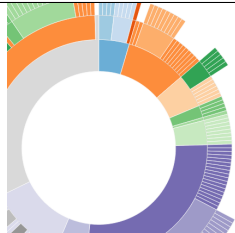
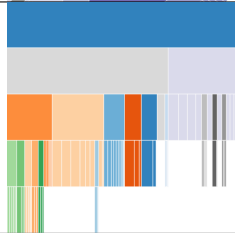
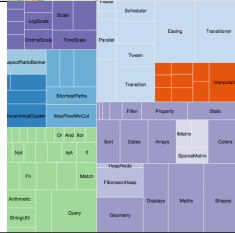
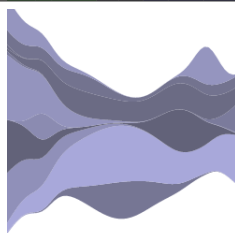
Name	Thumbnail	Description
Bar Chart		1 Dimension (bar) 1 Measure (bar height) Optionally: Picking interaction (click on bar)
Pie Chart		1 Dimension (slice) 1 Measure (slice size) Optionally: Picking interaction (click on slice)
Scatter Plot		1 Dimension (mark) 2 Measures (X, Y position) Optionally: +1 Dimension (symbol) +1 Measure (size) +1 Measure (color) Brushing interaction (2D)
Parallel Coordinates		1 Dimension (polyline) N Measures (one for each parallel axis) Optionally: Brushing interaction for each axis
Choropleth Map		1 Dimension, Space (geographic region) 1 Measure (color) Optionally: Small multiples visualization overlay Picking interaction (click on region) Pan & Zoom interaction

Table 5.2. Reusable Data Cube Visualizations (continued).

Name	Thumbnail	Description
Sunburst		1 Dimension, Hierarchical (slice) 1 Measure (slice size) Optionally: Picking interaction (click on slice)
Icicle		1 Dimension, Hierarchical (rectangles) 1 Measure (rectangle length) Optionally: Picking interaction (click on rectangle)
TreeMap		1 Dimension, Hierarchical (rectangles) 1 Measure (rectangle area) Optionally: Picking interaction (click on rectangle)
Streamgraph		2 Dimensions, (Time, streams) 1 Measure (stream thickness) Optionally: Picking interaction (click on stream), or Picking interaction (click on time), or Brushing interaction (across Time)

tion). The original data cube included the Time dimension as well, which effectively gets eliminated by taking only the 2010 slice as input. If historical data were available for each year, a line chart could be paired with the linked visualization to visualize and navigate the Time dimension.

The Religious Landscape data set can also be visualized as bar charts (or pie charts, or donut charts) overlaid on a geographic map. The bar charts can be squares with area representing population. Each bar within the square represents a religion. By pairing size of the bar chart with population, area corresponds to quantity of people accurately. In other words, each pixel on the screen inside a bar represents a fixed number of people. Visually, this makes sense because the human visual system naturally experiences and assesses sizes of things based on their area. [143] Pie charts could also be overlaid on a geographic map to express the same information.

Whether pie charts or bar charts, the embedded visualizations are at risk of occluding each other when plotted directly centered at the centroid of their corresponding regions. This can be overcome by using a force directed layout that repels embedded visualizations away from one another such that each is fully shown and not occluded by others. A similar technique has been used previously in cartography to produce cartograms based on circles (Dorling Cartogram) and squares (Demers Cartogram) [141].

Note that the geographic visualization overlay presents the same information as the above described linked map and bar chart interactive visualization. Rather than clicking on a country, the user can now just look at the center of any given country on the map to assess its religious breakdown. In this way, the geographically overlaid visualizations communicate more information at a time than the linked views.

The geographic bar chart visualization could also be augmented by a line chart if historical data is also available. The line chart could visualize the Time Dimension as a streamgraph chart showing global religious breakdown over time. Picking

interaction for years in the streamgraph can define the Time slice used as input to the other visualizations. This is one example of the general case that adding another visualization with picking interaction can effectively add another dimension to the data space users are capable of navigating with the visualization interface.

DBPedia is a knowledge graph representation of Wikipedia. DBPedia data is represented using RDF (Resource Description Framework), the World Wide Web Consortium standard knowledge graph representation for the Semantic Web [90]. In DBPedia, each Wikipedia Page about an entity is classified according to the DBPedia Ontology [17]. Resources in DBPedia can be aggregated along the DBPedia Ontology to produce a data cube that can be visualized. For example, one could visualize the count of resources for each ontology class using a Sunburst or Icicle visualization. Since resources have (lat, long) coordinates and associated dates, they can also be aggregated along Space and Time.

OpenStreetMap (OSM) is a community maintained crowdsourced geographic data repository [64]. The OSM database contains entities of numerous types including roads, points of interest, educational institutions, churches, buildings, military bases, shops, and public transit stations [33]. Any of these types can be counted across geographic space and used as indicators. For example, the number of buildings can be used to approximate population at a high resolution. Map features can also be aggregated by type and visualized to show the breakdown of the database.

The 2014 global Ebola outbreak is being monitored closely. Structured data about the outbreak is available and updating every day. This data provides measures “number of Ebola deaths” and “number of Ebola cases” for each affected West African nation for each day since the outbreak began. The New York Times published a graphical piece including a small multiples visualization of Ebola [140]. This is a topic prominent in the news today. A dynamically updating visualization for this data would be indispensable for journalists and decision makers alike.

How many people have been infected in Africa?

More than 8,000 people in Guinea, Liberia, Nigeria, Senegal and Sierra Leone have contracted Ebola since March, according to the World Health Organization, making this the biggest outbreak on record. [More than 3,800 people have died.](#)

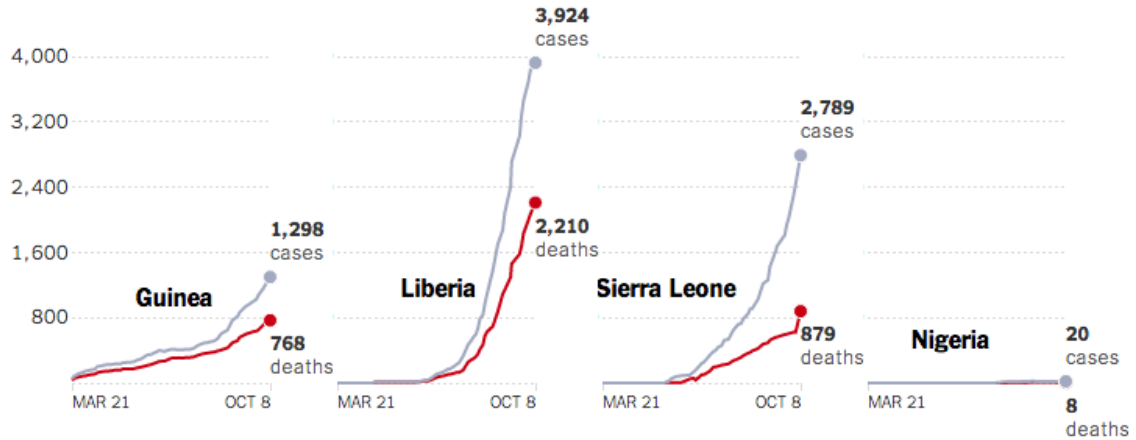


Figure 5.1. A small multiples line chart visualization of the 2014 Ebola outbreak in West Africa published by the New York Times [140]. This visualization shows the number of Ebola cases and deaths by day by region.

The Global Terrorism Database provides historical data about terrorist activities such as attacks with their time, place, and number of deaths. With daily news on ISIL (Islamic State of Iraq and the Levant), this live updating data set is a valuable resource for many people including journalists, policy makers, or anyone interested in learning about terrorist related events. This data can be aggregated into a data cube covering dimensions Time (nested structure of years, months, days), Space (geographic region), and providing the measure of number of deaths. This data could be visualized as a linked line chart and choropleth map. Picking in the line chart slices data used as input to the choropleth map. Zooming in the choropleth map dices the data shown in the line chart. This is similar to the linked view visualization of population described in figure 4.12.

Crowdsourcing may also be a promising approach for collecting and curating data. Each combination of region, time and measure can be formulated as an English ques-

tion, such as “What was the GDP of India in 1970?”. Crowdsourcing platforms such as Amazon Mechanical Turk and Crowdfunder can potentially be utilized to collect data. Workers may also be asked to enter the URL from which they found the data value. These URLs can be used to glean the structure of what is covered by various data sources, which can in turn be used to suggest URLs to future workers for certain regions of the data cube structure. Using this approach, it may be possible to develop an automated service that collects requested data on demand. Crowdsourcing techniques can also be used to refine the quality of the data collected.

5.5 A Grammar of Graphics Approach

A system based on named visualizations (e.g. “Scatter Plot”, “Bar Chart”) allows each visualization to have their own implementation, and does work. However, this approach is limited in that it does not take advantage of the deeper structure of visualizations. Leland Wilkinson’s “Grammar of Graphics” describes a language for expressing many types of visualizations using a single grammar [155]. Hadley Wickham has implemented this grammar and extended it in the R package `ggplot2` [154, 153]. A JSON-based grammar of graphics called VisJSON has also been developed [24].

Incorporating a grammar of graphics implementation into the proposed visualization platform would allow configuration of powerful and complex linked visualizations using a minimally complex structure. Each visualization on a page could be described by a grammar of graphics expression, rather than invoking a module that implements a named visualization. This kind of system would be much more elegant. One additional benefit of this organization is that when new features such as interaction techniques are implemented in the grammar of graphics engine, all visualizations immediately have the new interaction available. In a named visualization system, the interaction would need to be implemented once per visualization module.

One path to a grammar of graphics based approach within the existing reactive visualization framework is to create a grammar based solution that can express incrementally more visualizations. In this sense, it is essential to first have developed a named visualization approach, then start on the grammar of graphics approach. For example, a grammar of graphics based solution can be introduced that first can generate a scatter plot, then a bar chart, then a pie chart, then a stacked bar chart. As the implementation grows in completeness, named visualizations can be incrementally replaced with expressions in the grammar.

5.6 Final Thoughts

The hoped for eventual impact of this work is to enable common everyday people to understand their world better through data visualization. By providing a basis on which complex interactive visualization systems can be built and tailored to the data at hand, this work paves the way for future researchers and developers to create visualizations that represent data clearly and can be understood by many. Whether seen on a news site, included in a corporate report, or developed for scientific research, the impact of interactive data visualizations on users is immense and immediate.

If over time more and more data sets are imported into the UDC framework, made available publicly, and visualized, the phenomenal world will be covered more and more thoroughly by the reach of digitized information and visual presentations of it. Data and visualizations about various aspects of the world will become ever more commonplace, and part of common knowledge. With the advent of social media, sharing visualizations over the Web allows them to reach huge audiences very quickly. As time passes, select visualizations may become iconic, such as Minard's map of Napoleon's march has already. Such visualizations could eventually become "required reading" for academic courses of study. As such they could be integrated as interactive visualizations within E-Books being consumed via tablets or laptops.

There is a concept called “Mirror World” envisioned by Computer Science Professor David Gelerntner in which computers attain a kind of digital mirror of reality [59]. This dissertation introduces a means to define and populate a mirror world in terms of data cube aggregates, and also introduces a means to present the resulting data based view of the world via interactive visualizations. The hierarchical regions of Time and Space delineated by humans serve as the skeleton for our mirror world, and any measurable quantitative value that varies across Time, Space and additional dimensions can be incorporated into this mirror world. As more data becomes available, shedding digital light on the mysterious analog world, people will be able to understand the world around them more clearly through data visualization.

APPENDIX A

PSEUDOCODE CONVENTIONS

Throughout this document, pseudocode is used to express data structures and algorithms. The pseudocode used is similar to that found in the book “Introduction to Algorithms” [36], but differs significantly in that it uses a functional style. Primitive types in our pseudocode include numbers, strings, booleans, arrays, objects and functions. The following examples demonstrate the features of this pseudocode language.

```
1   $x = 5$ 
```

Line numbers appear to the left of each line of pseudocode. Variables can have any name comprised of characters without spaces, and can be assigned a value with the = symbol. Variables need not be explicitly declared. The scope of a variable is determined by where it is first assigned. Our pseudocode uses block scope, meaning that every indentation level introduces a new nested scope. On line 1 of the above pseudocode example, the variable x is defined and assigned the value of 5, a numeric literal.

The following pseudocode demonstrates numbers, strings and booleans.

```
1  myNumber = 5
2  myString = 'test'
3  myBoolean = TRUE
4  myOtherBoolean = FALSE
```

All numbers are treated as double precision floating point. Numeric literals in pseudocode become numbers (see line 1). String literals are denoted by single quotes and a monospace font (see line 2). Booleans can be either true or false. True and false are builtin constant boolean values denoted by all capitalized words (see lines 3 and 4). Camel case names starting with a lower case letter are used for most variables in our pseudocode.

```
1  add =  $\lambda(a, b)$ 
2      return  $a + b$ 
3  result = add(4, 6) // result is assigned the value 10
4  triple =  $\lambda(x)$  return  $x * 3$ 
5  triple(3) // evaluates to 9
```

The above pseudocode demonstrates how a function is defined and invoked, and also introduces comments. This example defines a function called *add* (on lines 1 and 2) that adds two numbers together. The λ (lambda) symbol defines a new anonymous function. Variables can be assigned functions as values using `=`. The comma separated names in parentheses directly following the λ are the arguments to the function. The pseudocode on lines following the λ that is indented one level constitutes the function body (also called the function closure). The function arguments are only visible inside the function closure.

Functions can be invoked using parentheses. The argument values are passed to the function in a comma separated list within parentheses. On line 3, the *add* function is invoked, passing the value 4 as argument *a* and 6 as argument *b*. The value returned by the function is assigned to the variable *result*. The function invocation causes the function body to execute, which adds the two numbers together and returns the resulting number using the “return” keyword on line 2. Lines 4 and 5 demonstrate that a simple anonymous function can be defined in a single line. Text following the `//` symbol is a comment, and is not executed.

```

1  myArray = []
2  myArray.push(5) // myArray now contains [5]
3  myArray.push(7) // myArray now contains [5, 7]
4  myArray.push(9) // myArray now contains [5, 7, 9]
5  myArray[0] // evaluates to 5
6  myArray[2] // evaluates to 9
7  myArray[1] = 3 // myArray now contains [5, 3, 9]
8  myBooleanArray = [TRUE, FALSE, TRUE, TRUE]
9  myStringArray = ['foo', 'bar']
10 numberOfBooleans = myBooleanArray.length // evaluates to 4
11 numberOfStrings = myStringArray.length // evaluates to 2
12 for str ∈ myStringArray
13   log(str) // prints 'foo' then 'bar'
14 myArray.map(triple) // evaluates to [15, 21, 27]

```

The above pseudocode demonstrates arrays. Arrays are ordered lists of elements. Arrays can contain elements of any type. Array literals are denoted by square brackets and can be empty (as in line 1) or populated (as in lines 8 and 9). Arrays have a built-in function attached to them called *push*, which appends a new element to the end of the array. Lines 2-4 demonstrate how *push* can be used to append items to an array. The dot notation seen on lines 2-4, 10-11 and 14 is used on arrays only to access the following built-in array functions and properties.

- *length* the number of items in the array
- *push(item)* appends an item to the end of an array
- *map(callback)* calls *callback(item)* for each *item* in the array

Square brackets denote access of array elements by index when placed directly after the array variable. Lines 5 and 6 demonstrate how square bracket notation can

be used to access values in an array based on their index. Line 7 demonstrates that square bracket notation can also be used to assign to values in an array. Lines 10 and 11 demonstrate the built-in property *length*, the number of elements in the array. Array indices start at zero.

Line 12 introduces the for loop construct. A for loop iterates over each element in the array. The indented code block following the for loop construct is executed once for each item in the array. In this example, each item is bound to the variable *str*, which is only visible within the for loop body. Line 13 invokes the built-in function *log(message)*, which prints out the message passed into it to.

Line 14 introduces the map construct. The built-in *map(iterator)* function applies the given *iterator(item)* function to each item in the array, and returns a new array populated with the returned values from *iterator*. In this example, the function *triple* defined earlier is applied to each element in the *myArray* array, yielding a new array with all values tripled.

```
1  myObject = {}
2  myObject.first = 'John' // myObject now contains {first : 'John'}
3  myObject['last'] = 'Doe' // now {first : 'John', last : 'Doe'}
4  myOtherObject = {first : 'Jane', last : 'Doe'}
5  box =
6    x : 50
7    y : 60
8    width : 100
9    height : 150
10 keys = keys(box) // evaluates to [x,y,width,height]
11 values = keys.map(λ(property) return box[property])
12 // values is assigned [50, 60, 100, 150]
13 box['nonexistentProperty'] // evaluates to NIL
```

The above pseudocode introduces objects. Objects are key-value mappings (sometimes called *maps* or *dictionaries*). Curly braces denote single-line object literals. Line 1 assigns the variable *myObject* to an empty object. Object properties can be assigned using dot notation, as in line 2, or square bracket notation, as in line 3. Bracket notation is useful when the property name is stored as a string in a variable. Object literals can contain key-value pairs denoted by *key : value* as in line 4. When an object literal spans multiple lines, the curly braces are omitted and the key-value pairs are indented, as in lines 5-9. Line 10 introduces the built-in function *keys*, which evaluates the keys of an object into an array. Line 11 demonstrates how the values of an object can be extracted into an array using the array *map* construct. A special value *NIL* is returned when attempting to access nonexistent object properties, as in line 13.

Our pseudocode assumes a single threaded execution environment with a built-in event loop, which may be implemented using the reactor pattern [126]. The event loop can be used to queue functions to be executed in the future. In our pseudocode, the *run* built-in function provides access to the event loop. Calling *run* and passing a function queues that function to be invoked in the future, after the current codepath terminates and all previously queued functions finish executing.

```

1  run( $\lambda()$  log('b'))
2  log('a')
3  // Prints a, then b

```

In the above pseudocode, line 1 queues an anonymous function that prints 'b' to run later, after the current codepath completes. While still inside the codepath which queued the function, line 2 prints 'a'. After line 2 executes, the current codepath terminates, causing the system to invoke queued function that prints b.

APPENDIX B

OPEN SOURCE PROJECTS

B.1 Ph. D.

<https://github.com/curran/phd>

This repository contains the LaTeX source code for the proposal and dissertation documents. This repository also contains source code for an early stage prototype of the end-to-end system, including reactive models, nested box layout, several visualizations and a configuration state runtime engine. These first prototypes were re-written and evolved to form the ModelJS and Model-Contrib projects.

B.2 ModelJS

<https://github.com/curran/model>

ModelJS was created in April 2014 as a simple proof of concept implementation of reactive models. The project arose out of a recurring pattern in my data visualization work at Rapid7 based on D3 and BackboneJS. ModelJS provides simple models akin to JavaScript objects that expose a **when** method that implements eventual resolution of reactive data flow graphs. The data flow graphs initiate an asynchronous breadth-first graph traversal algorithm when model values are changed. The JavaScript event loop is utilized as the queue in the breadth-first traversal. Multiple model changes that occur within the span of a single tick of the JavaScript event loop are batched together into a single invocation of dependent callback functions passed into the **when** method.

model-contrib

The place for open source contributions built with [ModelJS](#)

Modules

[barChart](#)
[lineChart](#)
[reactivis](#)
[scatterPlot](#)
[table](#)

Examples

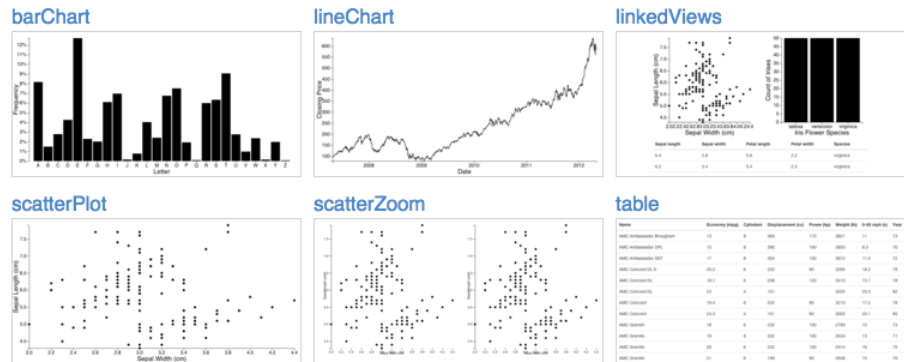


Figure B.1. The model-contrib project home page, showing a listing of reusable modules on the left and a gallery of examples with screenshots for each on the right.

B.3 Model-Contrib

<https://github.com/curran/model-contrib>

The Model-Contrib project is a playground for creating visualizations and reusable components using ModelJS. ModelJS contains the essence of reactive models, and all other contributions related to ModelJS belong in Model-Contrib. This includes example visualizations, visualizations with linked views, reusable visualization components. Other components such as Crossfilter integration and arbitrary SQL transformations are in Model-Contrib.

Model-Contrib includes an example viewer framework. This framework is responsible for displaying example screenshot thumbnails on the main model-contrib page, found at <http://curran.github.io/model-contrib/#/>, and also for rendering the example detail pages. The example detail pages run the example inside the page using an iFrame, render the README Markdown to HTML, and display the syntax-highlighted code for each source file using CodeMirror. AngularJS is used to

implement the example viewer client. PhantomJS and GraphicsMagick are used for generating the thumbnail images.

B.4 Reactivis

<https://github.com/curran/reactivis>

The Reactivis project contains reusable reactive graphs that can be composed together to create reusable visualization components. In this project, there is also an implementation of reactive data flow graph computation and visualization.

B.5 Overseer

<https://github.com/curran/overseer>

The Overseer project implements the runtime engine for dynamic configuration of visualizations. This project contains an implementation of the algorithm that computes the difference between two consecutive configuration states.

B.6 UDC

<https://github.com/curran/udc>

The UDC repository contains a JavaScript implementation of the Universal Data Cube API. This API is capable of loading, integrating and querying data sets represented in the UDC format. In the UDC format, each data sets consists of a data table (CSV file) and a metadata (JSON file).

B.7 UDC-Data

<https://github.com/curran/udc-data>

This repository contains example data sets represented using the UDC format. These data sets include the United Nations population data set, and the World Bank GDP data set.

BIBLIOGRAPHY

- [1] 20/20, Beyond. Product: Web data server. <http://www.beyond2020.com/index.php/data-solutions/products/web-data-server>, February 2014.
- [2] Abram, Greg, and Treinish, Lloyd. An extended data-flow architecture for data analysis and visualization. In *Proceedings of the 6th conference on Visualization'95* (1995), IEEE Computer Society, p. 263.
- [3] Agency, US Central Intelligence. Country comparison :: Gdp - real growth rate. <https://www.cia.gov/library/publications/the-world-factbook/rankorder/2003rank.html>, February 2014.
- [4] Agrawal, Rakesh, Gupta, Ashish, and Sarawagi, Sunita. Modeling multidimensional databases. In *Data Engineering, 1997. Proceedings. 13th International Conference on* (1997), IEEE, pp. 232–243.
- [5] Aizawa, Akiko, and Oyama, Keizo. A fast linkage detection scheme for multi-source information integration. In *Web Information Retrieval and Integration, 2005. WIRI'05. Proceedings. International Workshop on Challenges in* (2005), IEEE, pp. 30–39.
- [6] Anderson, Edgar. The irises of the gaspe peninsula. *Bulletin of the American Iris society* 59 (1935), 2–5.
- [7] Anselin, Luc. Interactive techniques and exploratory spatial data analysis. *Geographical Information Systems: principles, techniques, management and applications 1* (1999), 251–264.
- [8] Anselin, Luc, Syabri, Ibnu, and Smirnov, Oleg. Visualizing multivariate spatial correlation with dynamically linked windows. *Urbana* 51 (2002), 61801.
- [9] Bank, The World. Climate data api. <http://data.worldbank.org/developers/climate-data-api>, October 2014.
- [10] Becker, Richard A, and Cleveland, William S. Brushing scatterplots. *Technometrics* 29, 2 (1987), 127–142.
- [11] Berners-Lee, Tim, Hendler, James, Lassila, Ora, et al. The semantic web. *Scientific american* 284, 5 (2001), 28–37.
- [12] Berthold, Michael R, Cebon, Nicolas, Dill, Fabian, Gabriel, Thomas R, Kötter, Tobias, Meinl, Thorsten, Ohl, Peter, Sieb, Christoph, Thiel, Kilian, and Wiswedel, Bernd. *KNIME: The Konstanz information miner*. Springer, 2008.

- [13] Bertin, Jacques. Semiology of graphics: diagrams, networks, maps.
- [14] Bizer, Chris, Cyganiak, Richard, and Heath, Tom. How to publish linked data on the web. *Retrieved June 20* (2007), 2008.
- [15] Bizer, Christian, and Cyganiak, Richard. D2r server-publishing relational databases on the semantic web. In *5th international Semantic Web conference* (2006), p. 26.
- [16] Bizer, Christian, Heath, Tom, and Berners-Lee, Tim. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)* 5, 3 (2009), 1–22.
- [17] Bizer, Christian, Lehmann, Jens, Kobilarov, Georgi, Auer, Sören, Becker, Christian, Cyganiak, Richard, and Hellmann, Sebastian. Dbpedia-a crystallization point for the web of data. *Web Semantics: science, services and agents on the world wide web* 7, 3 (2009), 154–165.
- [18] Blaschka, Markus, Sapia, Carsten, Hofling, Gabriele, and Dinter, Barbara. Finding your way through multidimensional data models. In *Database and Expert Systems Applications, 1998. Proceedings. Ninth International Workshop on* (1998), IEEE, pp. 198–203.
- [19] Bostock, Michael, Ogievetsky, Vadim, and Heer, Jeffrey. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2011).
- [20] Bostock, Mike. Margin convention. <http://bl.ocks.org/mbostock/3019563>, June 2012.
- [21] Bostock, Mike, and Contributors. D3 gallery. <https://github.com/mbostock/d3/wiki/Gallery>, August 2014.
- [22] Bostock, Mike, Davies, Jason, and Contributors, D3. Tree layout. <https://github.com/mbostock/d3/wiki/Tree-Layout>, October 2014.
- [23] Boukhelifa, Nadia, Roberts, Jonathan C, and Rodgers, Peter. A coordination model for exploratory multi-view visualization. In *Proceedings of the International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2003)* (2003), IEEE COMPUTER SOC, 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA, pp. 76–85.
- [24] Brunssen, Vince C., and Malaika, Susan. Vizjson: The grammar of graphics in json. <http://www.ibm.com/developerworks/library/bd-vizjson/>, December 2013.
- [25] Card, Stuart K, Mackinlay, Jock D, and Shneiderman, Ben. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

- [26] Carlis, John, and Maguire, Joseph. *Mastering Data Modeling*. Addison-Wesley Longman Publishing Co., Inc., 2000.
- [27] Chaudhuri, Surajit, and Dayal, Umeshwar. An overview of data warehousing and olap technology. *ACM Sigmod record* 26, 1 (1997), 65–74.
- [28] Cho, Eunjoon, Myers, Seth A, and Leskovec, Jure. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining* (2011), ACM, pp. 1082–1090.
- [29] Christen, Peter. Febrl-: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), ACM, pp. 1065–1068.
- [30] Clifford, James, and Tansel, Abdullah Uz. On an algebra for historical relational databases: two views. In *ACM SIGMOD Record* (1985), vol. 14, ACM, pp. 247–265.
- [31] Codd, Edgar F, Codd, Sharon B, and Salley, Clynch T. Providing olap (on-line analytical processing) to user-analysts: An it mandate. *Codd and Date 32* (1993).
- [32] Contributors, BaconJS Project. Bacon.js: A small functional reactive programming lib for javascript. <https://github.com/baconjs/bacon.js/>, September 2014.
- [33] Contributors, OpenStreetMap. Map features. http://wiki.openstreetmap.org/wiki/Map_Features, October 2014.
- [34] Contributors, RxJS Project. Rxjs: The reactive extensions for javascript. <https://github.com/Reactive-Extensions/RxJS>, September 2014.
- [35] Cormack, Gordon V. A counterexample to the distributed operational transform and a corrected algorithm for point-to-point communication. *University of Waterloo Technical Report* (1995).
- [36] Cormen, Thomas H, Leiserson, Charles E, Rivest, Ronald L, Stein, Clifford, et al. *Introduction to algorithms*. MIT press, 2009.
- [37] Crockford, Douglas. The application/json media type for javascript object notation (json). <http://tools.ietf.org/html/rfc4627>, 2006.
- [38] Cuzzocrea, Alfredo, and Mansmann, Svetlana. Olap visualization: models, issues, and techniques. *Encyclopedia of Data Warehousing and Mining*, (2009), 1439–1446.

- [39] Cyganiak, Richard, Field, Simon, Gregory, Arofan, Halb, Wolfgang, and Tennison, Jeni. Semantic statistics: Bringing together sdmx and scovo. *LDOW 628* (2010).
- [40] Cyganiak, Richard, Reynolds, Dave, and Tennison, Jeni. The rdf data cube vocabulary. Tech. Rep. <http://www.w3.org/TR/vocab-data-cube>, W3C, December 2013.
- [41] Dabbish, Laura, Stuart, Colleen, Tsay, Jason, and Herbsleb, Jim. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work* (2012), ACM, pp. 1277–1286.
- [42] Dahlstrm, Erik, Dengler, Patrick, Grasso, Anthony, Lilley, Chris, McCormack, Cameron, Schepers, Doug, Watt, Jonathan, Ferraiolo, Jon, Jun, Fujisawa, and Jackson, Dean. Scalable vector graphics (svg) 1.1 (second edition). *W3C Recommendation* (2011).
- [43] Datta, Anindya, and Thomas, Helen. The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses. *Decision Support Systems* 27, 3 (1999), 289–301.
- [44] Deacon, John. Model-view-controller (mvc) architecture. *Online*[Citado em: 10 de março de 2006.] <http://www.jdl.co.uk/briefings/MVC.pdf> (2009).
- [45] Ding, Li, Shinavier, Joshua, Shangguan, Zhenning, and McGuinness, Deborah L. Sameas networks and beyond: analyzing deployment status and implications of owl: sameas in linked data. In *The Semantic Web–ISWC 2010*. Springer, 2010, pp. 145–160.
- [46] Doan, AnHai, Domingos, Pedro, and Halevy, Alon Y. Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM Sigmod Record* (2001), vol. 30, ACM, pp. 509–520.
- [47] Doan, AnHai, Domingos, Pedro, and Levy, Alon Y. Learning source description for data integration. In *WebDB (Informal Proceedings)* (2000), pp. 81–86.
- [48] Doan, AnHai, Halevy, Alon, and Ives, Zachary. *Principles of data integration*. Elsevier, 2012.
- [49] Doan, AnHai, and Halevy, Alon Y. Semantic integration research in the database community: A brief survey. *AI magazine* 26, 1 (2005), 83.
- [50] Eick, Stephen G. Visualizing multi-dimensional data. *ACM SIGGRAPH computer graphics* 34, 1 (2000), 61–67.
- [51] Elfeky, Mohamed G, Verykios, Vassilios S, and Elmagarmid, Ahmed K. Tailor: A record linkage toolbox. In *Data Engineering, 2002. Proceedings. 18th International Conference on* (2002), IEEE, pp. 17–28.

- [52] Elliott, Conal, and Hudak, Paul. Functional reactive animation. In *ACM SIGPLAN Notices* (1997), vol. 32, ACM, pp. 263–273.
- [53] Fagin, Ronald, Kolaitis, Phokion G, Miller, Renée J, and Popa, Lucian. Data exchange: Semantics and query answering. In *Database TheoryICDT 2003*. Springer, 2003, pp. 207–224.
- [54] for Disease Control, US Centers. National vital statistics system. <http://205.207.175.93/Vitalstats/TableViewer/tableView.aspx>, February 2014.
- [55] Foundation, National Science. Science and engineering indicators 2012 state data tool. <http://www.nsf.gov/statistics/seind12/c8/interactive/map.cfm?table=31&year=2009>, February 2014.
- [56] Fulton, Steve, and Fulton, Jeff. *HTML5 Canvas*. O’Reilly Media, 2013.
- [57] Gallo, Giorgio, and Pallottino, Stefano. Shortest path algorithms. *Annals of Operations Research* 13, 1 (1988), 1–79.
- [58] Gapminder. Data in gapminder world. <http://www.gapminder.org/data/>, February 2014.
- [59] Gelernter, David Hillel. *Mirror worlds, or, The day software puts the universe in a shoebox—: how it will happen and what it will mean*. Oxford University Press New York, 1991.
- [60] Gonzalez, Hector, Halevy, Alon Y, Jensen, Christian S, Langen, Anno, Madhavan, Jayant, Shapley, Rebecca, Shen, Warren, and Goldberg-Kidon, Jonathan. Google fusion tables: web-centered data management and collaboration. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (2010), ACM, pp. 1061–1066.
- [61] Gray, Jim, Chaudhuri, Surajit, Bosworth, Adam, Layman, Andrew, Reichart, Don, Venkatrao, Murali, Pellow, Frank, and Pirahesh, Hamid. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery* 1, 1 (1997), 29–53.
- [62] Gu, Lifang, Baxter, Rohan, Vickers, Deanne, and Rainsford, Chris. Record linkage: Current practice and future directions. *CSIRO Mathematical and Information Sciences Technical Report 3* (2003), 83.
- [63] Gyssens, Marc, and Lakshmanan, Laks VS. A foundation for multi-dimensional databases. In *VLDB* (1997), vol. 97, pp. 106–115.
- [64] Haklay, Mordechai, and Weber, Patrick. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE* 7, 4 (2008), 12–18.
- [65] Halbwachs, Nicholas, Caspi, Paul, Raymond, Pascal, and Pilaud, Daniel. The synchronous data flow programming language lustre. *Proceedings of the IEEE* 79, 9 (1991), 1305–1320.

- [66] Halevy, Alon, Rajaraman, Anand, and Ordille, Joann. Data integration: the teenage years. In *Proceedings of the 32nd international conference on Very large data bases* (2006), VLDB Endowment, pp. 9–16.
- [67] Halpin, Harry, Hayes, Patrick J, McCusker, James P, McGuinness, Deborah L, and Thompson, Henry S. When owl: sameas isnt the same: An analysis of identity in linked data. In *The Semantic Web-ISWC 2010*. Springer, 2010, pp. 305–320.
- [68] Hanrahan, Pat, Stolte, Chris, and Mackinlay, Jock. visual analysis for everyone. *Tableau White paper 4* (2007).
- [69] Hatanaka, Iwao, and Hughes, Stephen C. Providing multiple views in a model-view-controller architecture, July 20 1999. US Patent 5,926,177.
- [70] Hauser, Helwig, Ledermann, Florian, and Doleisch, Helmut. Angular brushing of extended parallel coordinates. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on* (2002), IEEE, pp. 127–130.
- [71] He, Bin, and Chang, Kevin Chen-Chuan. Statistical schema matching across web query interfaces. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (2003), ACM, pp. 217–228.
- [72] Heer, Jeffrey, and Agrawala, Maneesh. Software design patterns for information visualization. *Visualization and Computer Graphics, IEEE Transactions on* 12, 5 (2006), 853–860.
- [73] Hickson, Ian, Berjon, Robin, Faulkner, Steve, Leithead, Travis, Navara, Erika Doyle, O’Connor, Edward, and Pfeiffer, Silvia. Html5: A vocabulary and associated apis for html and xhtml. *W3C Working Draft edition* (2013).
- [74] Hils, Daniel D. Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing* 3, 1 (1992), 69–101.
- [75] Holman, CD’Arcy J, Bass, A John, Rouse, Ian L, and Hobbs, Michael ST. Population-based linkage of health records in western australia: development of a health services research linked database. *Australian and New Zealand journal of public health* 23, 5 (1999), 453–459.
- [76] Hudak, Paul, Courtney, Antony, Nilsson, Henrik, and Peterson, John. Arrows, robots, and functional reactive programming. In *Advanced Functional Programming*. Springer, 2003, pp. 159–187.
- [77] Inc., Google. Google public data explorer. <https://www.google.com/publicdata/directory>, September 2014.
- [78] Inc., TIBCO Software. Tibco spotfire. <http://spotfire.tibco.com/>, September 2014.

- [79] Institute, SAS. *SAS Visual Analytics 6. 1: User's Guide*. Sas Institute, 2012.
- [80] Jaro, Matthew A. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association* 84, 406 (1989), 414–420.
- [81] Jaro, Matthew A. Probabilistic linkage of large public health data files. *Statistics in medicine* 14, 5-7 (1995), 491–498.
- [82] Jérôme, Shvaiko, Pavel, et al. *Ontology matching*. Springer, 2007.
- [83] Kalfoglou, Yannis, and Schorlemmer, Marco. Ontology mapping: the state of the art. *The knowledge engineering review* 18, 1 (2003), 1–31.
- [84] Kämpgen, Benedikt, and Harth, Andreas. Transforming statistical linked data for use in olap systems. In *Proceedings of the 7th international conference on Semantic systems* (2011), ACM, pp. 33–40.
- [85] Kandel, Sean, Paepcke, Andreas, Hellerstein, Joseph, and Heer, Jeffrey. Wrangler: Interactive visual specification of data transformation scripts. In *PART 5—Proceedings of the 2011 annual conference on Human factors in computing systems* (2011), ACM, pp. 3363–3372.
- [86] Kang, Jaewoo, and Naughton, Jeffrey F. On schema matching with opaque column names and data values. In *SIGMOD Conference* (2003), pp. 205–216.
- [87] Keim, Daniel A. Information visualization and visual data mining. *Visualization and Computer Graphics, IEEE Transactions on* 8, 1 (2002), 1–8.
- [88] Kelleher, Curran. Modeljs open source project. <https://github.com/curran/model>, August 2014.
- [89] Kimball, Ralph. *The data warehouse lifecycle toolkit: expert methods for designing, developing, and deploying data warehouses*. Wiley. com, 1998.
- [90] Klyne, Graham, and Carroll, Jeremy J. Resource description framework (rdf): Concepts and abstract syntax.
- [91] Koudas, Nick, Sarawagi, Sunita, and Srivastava, Divesh. Record linkage: similarity measures and algorithms. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (2006), ACM, pp. 802–803.
- [92] Krasner, Glenn E, Pope, Stephen T, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming* 1, 3 (1988), 26–49.
- [93] Labs, Alpine Data. Open chorus foundation: An open platform for data collaboration in the enterprise. <http://openchorus.org/>, September 2014.

- [94] Leff, Avraham, and Rayfield, James T. Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International* (2001), IEEE, pp. 118–127.
- [95] Lenzerini, Maurizio. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2002), ACM, pp. 233–246.
- [96] Lerner, Reuven M. At the forge: twitter bootstrap. *Linux Journal* 2012, 218 (2012), 6.
- [97] Li, Chang, and Wang, X Sean. A data model for supporting on-line analytical processing. In *Proceedings of the fifth international conference on Information and knowledge management* (1996), ACM, pp. 81–88.
- [98] Lins, Lauro, Klosowski, James T, and Scheidegger, Carlos. Nanocubes for real-time exploration of spatiotemporal datasets. *Visualization and Computer Graphics, IEEE Transactions on* 19, 12 (2013), 2456–2465.
- [99] Liu, Zhicheng, and Heer, Jeffrey. The effects of interactive latency on exploratory visual analysis.
- [100] Liu, Zhicheng, Jiang, Biye, and Heer, Jeffrey. immens: Real-time visual querying of big data. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 421–430.
- [101] LLC, Wolfram Alpha. Wolfram alpha, Aug. 2014.
- [102] Lopez, Vanessa, Kotoulas, Spyros, Sbodio, Marco Luca, Stephenson, Martin, Gkoulalas-Divanis, Aris, and Mac Aonghusa, Pól. Queriocity: A linked data platform for urban information management. In *The Semantic Web-ISWC 2012*. Springer, 2012, pp. 148–163.
- [103] Lytras, Miltiadis D, and García, Roberto. Semantic web applications: a framework for industry and business exploitation—what is needed for the adoption of the semantic web from the market and industry. *International Journal of Knowledge and Learning* 4, 1 (2008), 93–108.
- [104] Maali, Fadi, Cyganiak, Richard, and Peristeras, Vassilios. A publishing pipeline for linked government data. In *The Semantic Web: Research and Applications*. Springer, 2012, pp. 778–792.
- [105] Mackinlay, Jock. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics (TOG)* 5, 2 (1986), 110–141.
- [106] Madhavan, Jayant, Bernstein, Philip A, and Rahm, Erhard. Generic schema matching with cupid. In *VLDB* (2001), vol. 1, pp. 49–58.

- [107] Madhavan, Jayant, Jeffery, S, Cohen, Shirley, Dong, X, Ko, David, Yu, Cong, and Halevy, Alon. Web-scale data integration: You can only afford to pay as you go. In *Proceedings of CIDR* (2007), pp. 342–350.
- [108] Mansmann, Svetlana, and Scholl, Marc H. Visual olap: A new paradigm for exploring multidimensional aggregates. In *Proc. of IADIS Intl Conf. on Computer Graphics and Visualization (CGV)* (2008), pp. 59–66.
- [109] Matsuda, Kouichi, and Lea, Rodger. *WebGL programming guide: interactive 3D graphics programming with WebGL*. Pearson Education, 2013.
- [110] Milo, Tova, and Zohar, Sagit. Using schema matching to simplify heterogeneous data translation. In *VLDB* (1998), vol. 98, Citeseer, pp. 24–27.
- [111] Nations, United. Millenium development goals indicators; country level data. <http://unstats.un.org/unsd/mdg/Data.aspx>, February 2014.
- [112] Nations, United. World population prospects: The 2012 revision. <http://esa.un.org/unpd/wpp/Excel-Data/population.htm>, February 2014.
- [113] Noy, Natalya F. Semantic integration: a survey of ontology-based approaches. *ACM Sigmod Record* 33, 4 (2004), 65–70.
- [114] Noy, Natalya F. Ontology mapping. In *Handbook on ontologies*. Springer, 2009, pp. 573–590.
- [115] Noy, Natalya F, and Musen, Mark A. The prompt suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies* 59, 6 (2003), 983–1024.
- [116] Osmani, Addy. *Learning JavaScript Design Patterns*. ” O’Reilly Media, Inc.”, 2012.
- [117] O’Sullivan, Bryan. Making sense of revision-control systems. *Communications of the ACM* 52, 9 (2009), 56–62.
- [118] Quilitz, Bastian, and Leser, Ulf. Querying distributed rdf data sources with sparql. In *The Semantic Web: Research and Applications*. Springer, 2008, pp. 524–538.
- [119] Rahm, Erhard, and Bernstein, Philip A. A survey of approaches to automatic schema matching. *the VLDB Journal* 10, 4 (2001), 334–350.
- [120] Ramakrishnan, Raghu, and Gehrke, Johannes. *Database management systems*. Osborne/McGraw-Hill, 2000.
- [121] Reingold, Edward M, and Tilford, John S. Tidier drawings of trees. *Software Engineering, IEEE Transactions on*, 2 (1981), 223–228.
- [122] Roberts, Jonathan C. Exploratory visualization with multiple linked views.

- [123] Rosling, Hans, Rosling, Rönnlund A, and Rosling, Ola. New software brings statistics beyond the eye. *Statistics, Knowledge and Policy: Key Indicators to Inform Decision Making*. Paris, France: OECD Publishing (2005), 522–530.
- [124] Royal, Cindy. The journalist as programmer: A case study of the new york times interactive news technology department. In *International Symposium on Online Journalism* (2010).
- [125] Salas, Percy E, Martin, Michael, Mota, Fernando Maia Da, Breitman, Karin, Auer, Sören, and Casanova, Marco A. Publishing statistical data on the web. In *Proceedings of 6th International IEEE Conference on Semantic Computing* (Palermo, Italy, 2012), IEEE 2012, IEEE.
- [126] Schmidt, Douglas C. Reactor: An object behavioral pattern for concurrent event demultiplexing and dispatching.
- [127] Scotch, Mathew, Parmanto, Bambang, and Monaco, Valerie. Usability evaluation of the spatial olap visualization and analysis tool (sovat). *Journal of Usability Studies* 2, 2 (2007), 76–95.
- [128] Scotch, Matthew, and Parmanto, Bambang. Sovat: Spatial olap visualization and analysis tool. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on* (2005), IEEE, pp. 142b–142b.
- [129] Shneiderman, Ben. Dynamic queries for visual information seeking. *Software, IEEE* 11, 6 (1994), 70–77.
- [130] Shneiderman, Ben. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on* (1996), IEEE, pp. 336–343.
- [131] Shvaiko, Pavel, and Euzenat, Jérôme. A survey of schema-based matching approaches. In *Journal on Data Semantics IV*. Springer, 2005, pp. 146–171.
- [132] Sifer, Mark. A visual interface technique for exploring olap data with coordinated dimension hierarchies. In *Proceedings of the twelfth international conference on Information and knowledge management* (2003), ACM, pp. 532–535.
- [133] Stolte, Chris, Tang, Diane, and Hanrahan, Pat. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *Visualization and Computer Graphics, IEEE Transactions on* 8, 1 (2002), 52–65.
- [134] Stolte, Chris, Tang, Diane, and Hanrahan, Pat. Query, analysis, and visualization of hierarchically structured data using polaris. In *KDD* (2002), vol. 2, Citeseer, pp. 112–122.
- [135] Stolte, Chris, Tang, Diane, and Hanrahan, Pat. Multiscale visualization using data cubes. *Visualization and Computer Graphics, IEEE Transactions on* 9, 2 (2003), 176–187.

- [136] Sumbaly, Roshan, Kreps, Jay, and Shah, Sam. The big data ecosystem at linkedin. In *Proceedings of the 2013 international conference on Management of data* (2013), ACM, pp. 1125–1134.
- [137] Sun, Chengzheng, and Ellis, Clarence. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work* (1998), ACM, pp. 59–68.
- [138] Sundram, Jason. A full stack approach to data visualization: Terabytes (and beyond) at facebook. https://www.youtube.com/watch?v=hGDNBGShQVY&list=PLl1gxAbM671YLcGj8M00_6X1B1Pg5eAGqG&index=18, April 2014.
- [139] Techapichetvanich, Kesaraporn, and Datta, Amitava. Interactive visualization for olap. In *Computational Science and Its Applications–ICCSA 2005*. Springer, 2005, pp. 206–214.
- [140] Times, The New York. Ebola facts: How many patients are being treated outside of west africa? <http://www.nytimes.com/interactive/2014/07/31/world/africa/ebola-virus-outbreak-qa.html>, October 2014.
- [141] Tobler, Waldo. Thirty five years of computer cartograms. *ANNALS of the Association of American Geographers* 94, 1 (2004), 58–73.
- [142] Tobler, Waldo, and Chen, Zi-tan. A quadtree for global information storage. *Geographical Analysis* 18, 4 (1986), 360–371.
- [143] Tufte, Edward R, and Graves-Morris, PR. *The visual display of quantitative information*, vol. 2. Graphics press Cheshire, CT, 1983.
- [144] Uschold, Michael, and Gruninger, Michael. Ontologies and semantics for seamless connectivity. *ACM SIGMod Record* 33, 4 (2004), 58–64.
- [145] Vassiliadis, Panos. Modeling multidimensional databases, cubes and cube operations. In *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on* (1998), IEEE, pp. 53–62.
- [146] Vassiliadis, Panos, and Sellis, Timos. A survey of logical models for olap databases. *ACM Sigmod Record* 28, 4 (1999), 64–69.
- [147] Viegas, Fernanda B, Wattenberg, Martin, Van Ham, Frank, Kriss, Jesse, and McKeon, Matt. Manyeyes: a site for visualization at internet scale. *Visualization and Computer Graphics, IEEE Transactions on* 13, 6 (2007), 1121–1128.
- [148] Wache, Holger, Voegelé, Thomas, Visser, Ubbo, Stuckenschmidt, Heiner, Schuster, Gerhard, Neumann, Holger, and Hübner, Sebastian. Ontology-based integration of information—a survey of existing approaches. In *IJCAI-01 workshop: ontologies and information sharing* (2001), vol. 2001, Citeseer, pp. 108–117.

- [149] Wan, Zhanyong, and Hudak, Paul. Functional reactive programming from first principles. In *ACM SIGPLAN Notices* (2000), vol. 35, ACM, pp. 242–252.
- [150] Ward, Matthew O. Xmdvtool: Integrating multiple methods for visualizing multivariate data. In *Proceedings of the Conference on Visualization'94* (1994), IEEE Computer Society Press, pp. 326–333.
- [151] Weaver, Chris. Building highly-coordinated visualizations in improvise. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on* (2004), IEEE, pp. 159–166.
- [152] White, Richard A, and Stemwedel, Sally W. The quadrilateralized spherical cube and quad-tree for all sky data. In *Astronomical Data Analysis Software and Systems I* (1992), vol. 25, p. 379.
- [153] Wickham, Hadley. *ggplot2: elegant graphics for data analysis*. Springer, 2009.
- [154] Wickham, Hadley. A layered grammar of graphics. *Journal of Computational and Graphical Statistics* 19, 1 (2010), 3–28.
- [155] Wilkinson, Leland. *The grammar of graphics*. Springer, 2005.
- [156] Winkler, William E. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau* (1999), Citeseer.
- [157] Winkler, William E. Overview of record linkage and current research directions. In *Bureau of the Census* (2006), Citeseer.
- [158] Yi, Ji Soo, ah Kang, Youn, Stasko, John T, and Jacko, Julie A. Toward a deeper understanding of the role of interaction in information visualization. *Visualization and Computer Graphics, IEEE Transactions on* 13, 6 (2007), 1224–1231.
- [159] Ziegler, Patrick, and Dittrich, Klaus R. Three decades of data integration-all problems solved? In *IFIP congress topical sessions* (2004), Springer, pp. 3–12.