Doctoral Dissertation

Curran Kelleher

5/2/2014

Abstract

There is immense potential value in data that is not being realized. While many data sets are available, it is difficult to realize their full value because they are made available using many different formats, protocols and vocabularies. The heterogeneity of formats, protocols and vocabularies makes it difficult to combine data sets together and hinders the development of reusable data visualization software. While it is straightforward to produce static visualizations of just about any data set by customizing existing examples, there is a lack of generalized visualization software that supports the creation of interactive visualizations and visualization dashboards with multiple linked views. The contribution of this dissertation is a collection of data structures and algorithms supporting integration and interactive visualization of many data sets using interactive visualization dashboards with multiple linked views. A proof of concept implementation demonstrates support for several public data sets and well known visualization techniques.

Contents

Chapter 1

Introduction

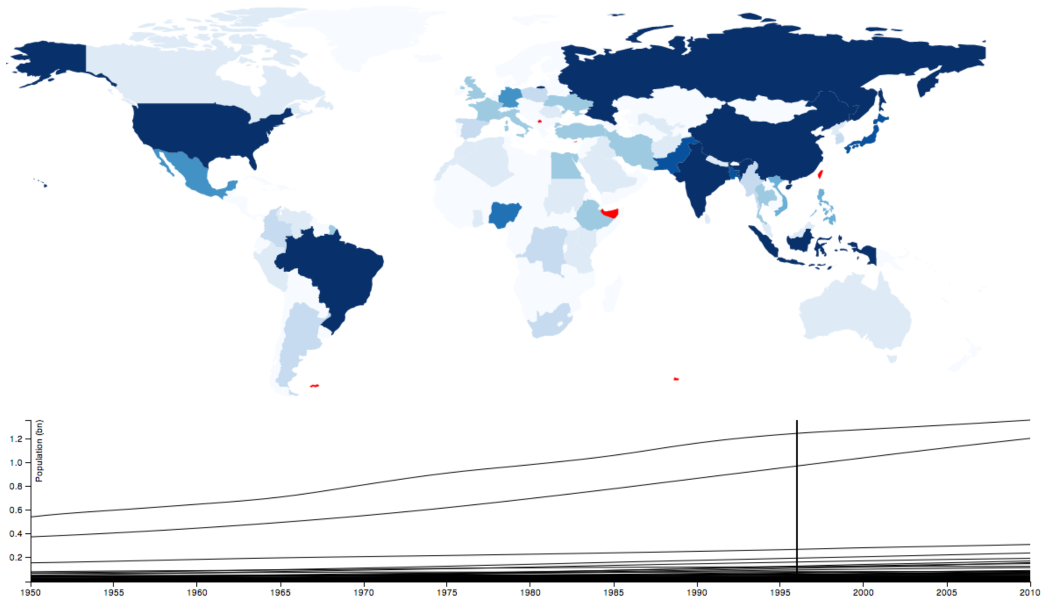
The contribution of this dissertation is a coherent collection of novel data structures and algorithms for integration and interactive visualization of many data sets from multiple sources, based on the data cube concept. The proposed data representation framework will allow data sets to be combined together and visualized using interactive visualization dashboards, giving users the sense that the data exists within a single unified structure. The framework is designed to be able to represent and integrate an arbitrary number of data sets created independently of one another, and expose the integrated structure to reusable visualizations. The reusable visualizations can be combined together in dashboard layouts with multiple linked views using existing interaction techniques such as brushing and linking. The proposed data representation and visualization framework is fundamentally new, and will allow heterogeneous data sets to be explored in a unified way that was never before possible.

The overall goal of this work is to build digital telescope into the universe of phenomena on Earth via publicly available data. For example, consider public 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 and present the data visually, a deeper understanding of the world can be gleaned through the lens of public data. The focus of this dissertation is on applications involving public data, however the techniques introduced can be applied to any data sets that can be conceptually modeled as data cubes, regardless of whether they are public or private.

Consider the data from the US Census that covers population statistics for US States from 1950 to 2010. Consider also population statistics from the United Nations covering World Countries from 1970 to 2012. These two data sets may use different identifiers for years and geographic regions, but they cover an overlapping conceptual data space of time, geography and population. From these two data sets it is possible to create a visualization dashboard with a map of the world showing population as color and a corresponding line graph showing population for each region as lines.

If the user views the whole world, the UN population data is shown for each country. If the user zooms into the US, US Census data is shown for each state. If the user selects a point of time in the line graph, the data shown on the map is from that point in time. If the user pans and zooms on the map, the lines in the line graph update to only show the regions visible on the map. This is one example of an interactive visualization dashboard with multiple linked views (the timeline and map views) operating over multiple data sets integrated from different sources (the United Nations population data and the US Census population data).

Figure 1.1: An example visualization dashboard with multiple linked views. Selecting a time on the line chart slices the data displayed on the choropleth map. Zooming and panning on the choropleth map filter the country profiles shown in the line chart. The vertical black line on the line chart indicates the selected year.



TODO Improve this visualization - add color legend, add names to lines

Data cubes, also known as OLAP (OnLine Analytical Processing) cubes, can represent data that contains measures aggregated (typically using sum or average) along categorical hierarchies. The data cube concept emerged from the field of data warehousing as a way to summarize transactional data, allowing analysts to get a bird's eye view of company activities. The term OLAP stands in contrast to the term 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.

The data cube concept and structure can be used to model existing data sets 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 or other dimensions such as gender, age range, ethnicity or industry sector. Any categorization scheme containing distinct entities, organized as an unorered collection, an ordered collection, or a hierarchy can be modeled as a dimension. Any numeric value that represents an aggregated statistical summary using sum, average, or other aggregation operator can be modeled as a measure.

With this approach, it is possible to model many data sets together using shared dimensions and measures. This allows integration of many data sets together in a single unified structure. Existing data cube technologies assume that data cubes will be computed from a relational source, and are not designed to handle integration of pre-computed data cubes that may use inconsistent identifiers for common dimensions and inconsistent scaling factors for common measures. 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 inconsistent scaling factors for common measures.

The data cube structure lends itself particularly well to visualization. Long standing data visualization theory presented by Bertin [7] and Mackinlay [64] identify effective ways to visually encode data based on data fields that are nominal, ordinal or quantitative. Visualization and interaction techniques have been explored for hierarchical (tree-based) data as well [36, 110]. Data cubes can contain data of all these types (dimensions can be nominal, ordered, or hierarchical; measures are quantitative). Therefore existing visualization theory can be applied to the data cube model to determine which visualizations are appropriate for representing which data, depending on its data cube structure.

1.1 Related Work

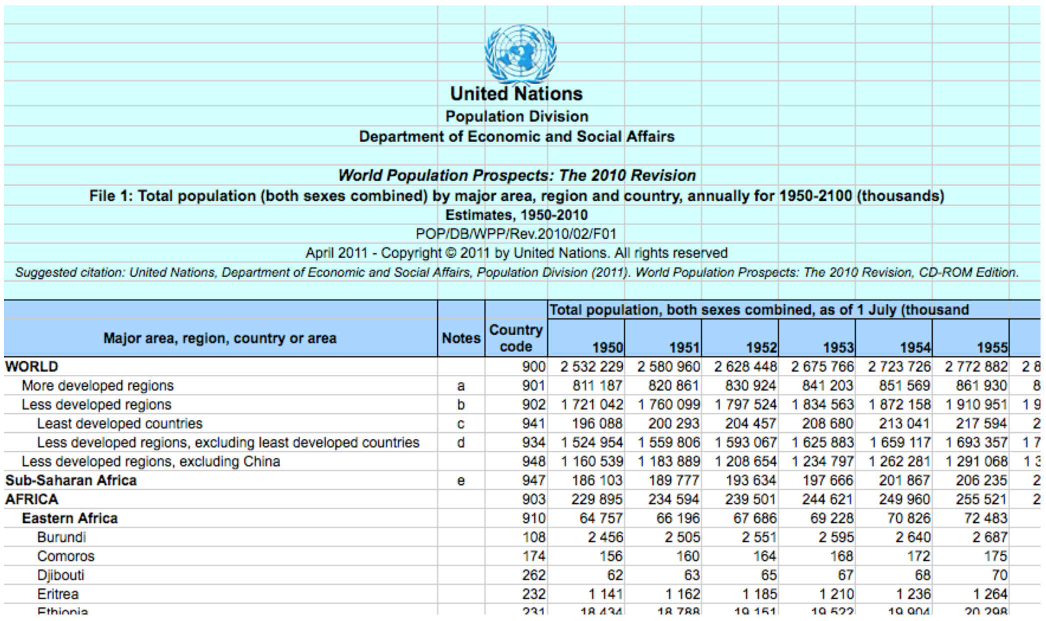
Previous work relating to this dissertation falls into four major categories:

- data representation - structures, models and formats for data
- data integration - merging data from many sources
- data visualization - transforming data into interactive graphics
- Web graphics technology - HTML5 graphics APIs and libraries

1.1.1 Data Representation

In today's world of "information overload", data takes many forms. Perhaps the most familiar data representation system today is Microsoft Excel, which is capable of representing data tables as well as complex operations across the data values. Many organizations use Excel to manage data or make data available as Excel spreadsheets. For example, the United Nations Department of Economic and Social Affairs makes their population statistics available in Excel format (see figure 1.2). Many data tables on the Web are made available in simpler text-based formats such as CSV (Comma Separated Values) or TSV (Tab Separated Values). Data sets are also frequently made available in proprietary formats that may be text based or binary, such as ESRI Shapefiles or Adobe PDF documents.

Relational database systems provide a mature data management solution and are widely adopted [82]. The relational model has well understood theo-



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). World Population Prospects: The 2010 Revision, CD-ROM Edition.

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	19 904	20 298	

Figure 1.2: The United Nations Population Prospects data set [73], made available in Excel format. This is an example of a public data set that can be imported into our data representation framework.

retical underpinnings such as relational algebra [16]. Data warehouse systems are typically built on the relational model and are augmented by multi-scale aggregated data structures called data cubes, also known as OLAP (OnLine Analytical Processing) cubes [37, 17]. Data cubes contain summaries of the collection of facts stored in a relational database [12]. 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 an abstraction that handles aggregation, they are a widely used method of data abstraction for supporting visualization and analysis tasks [93]. Kimball pioneered the area of “Dimensional Modeling”, which concerns constructing data warehouse schemas amenable to data cube construction and analysis [54]. 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 [25, 101, 100, 61, 2, 39, 8]. The data cube structure has also been used to model user Web browsing sessions to support data mining algorithms for Web prefetching [111].

1.1.2 Data Integration

The field of data integration offers many techniques for combining data from multiple sources based on the relational model [28] as well as from a theoretical perspective [60, 40, 113]. Schema matching is the area of data integration that concerns semantic matching between the attributes of data tables from different sources [81, 33]. Schema matching may be performed manually, however it must be automated in order to scale to hundreds or thousands of different sources. Numerous approaches for automated schema matching have been proposed [89, 26, 52, 72, 65, 27]. Schema matching approaches aimed specifically at Web and Ontology based data integration have also been proposed [42, 76, 29, 66, 50, 77, 99, 103, 78, 49]. Data matching (also known as record linkage) is the area of data integration focusing on resolving different identifiers to the same real-world entity [108, 109, 56, 4, 38, 107]. Record linkage has been applied extensively to public data [48, 47, 45]. Several tools have been introduced that aid users in data integration tasks via a graphical user interface [15, 51, 31].

1.1.3 Data Visualization

The field of information visualization offers many compelling approaches for visualizing data [53]. Arguably the first significant work concerning data visualization was William Playfair's "Commercial and Political Atlas", published in 1786 [80]. In this work, Playfair introduced the Bar Chart, Pie Chart, and Line Graph. The first attempt at a systematic formalization of data visualization was Jacques Bertin's "Semiology of Graphics" [7]. 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" [106] and Jock Mackinlay's APT (A Presentation Tool) system [64].

Interactive data visualization systems based on straightforward tables are plentiful. Tableau is a commercial visualization package supporting creation of interactive visualization dashboards [41]. Spotfire is a framework for constructing information visualization systems [3]. GGobi is an extensible framework (based on XGobi) for interactive data visualization based on the R platform [95, 94]. Gee et al. presented the Universal Visualization Platform, a general interactive visualization infrastructure upon which others can build knowledge discovery applications. North et al. introduced Snap-

Together Visualization, a user interface for coordinating visualizations [75] and an extension of the architecture for the Web [74]. Fekete introduced the InfoVis Toolkit, a collection of reusable visualization components. GeoDa is a tool for interactive visual exploration of multidimentional data with a geospatial component [5].

Much effort has been placed on generating taxonomies of visualization techniques. Chi et al. introduced a taxonomy of visualizations based on the Data State Reference Model [13, 14]. Shneiderman introduced a more general taxonomy based on tasks and data types [88]. Card et al. made steps toward characterizing the entire design space of data visualizations based on Bertin's theory [11]. Tufte explored numerous visualization techniques for quantitative information in general, many of which can be applied to visualization of data cubes [98].

Much work has been done regarding 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 [93, 92, 91]. In this work the authors introduce theoretical underpinnings of a visualization system capable of navigating hierarchical data cubes with a combination of data abstraction and visual abstraction. Eick introduced the ADVISOR system, which focused on visualizing data cubes and categorizing classes of data cube visualizations [30]. Eick discussed the following "perspectives" of data cube visualizations: Single Measure, Multiple Measures and Anchored Measures. He also discussed the following interaction techniques for data cube visualizations: Undo and Redo, Bookmarks, Selection (Replace, Intersect, Add, Subtract, Select All, Unselect All, Toggle), and Exclusion.

Mansmann coined the term “Visual OLAP”, framed it as a fundamentally new paradigm for exploring multidimensional aggregates [70], explored applications of hierarchical visualization techniques to OLAP cubes [69] and extended Visual OLAP to support irregular hierarchies [68]. Cuzzocrea et al. surveyed the area of data cube visualization in depth [20] and have made several contributions regarding semantics-aware OLAP visualization [22] and a hierarchy driven compression technique for OLAP visualization [21]. Scotch et al. developed and evaluated SOVAT, a Spatial OLAP visualization and analysis tool applied to community health assessments [86, 85]. Lee and Ong introduced a visualisation technique for knowledge discovery in OLAP combining elements of bar charts and parallel coordinates [58]. Maniatis et al. explored how OLAP cubes can be visualized using TableLens and other techniques [67]. Data cubes have also been utilized as the foundational data structure for several “Big Data” visualization systems [62, 63].

Interactions within data visualization environments have been well studied. Becker et al. investigated brushing in scatter plots [6]. Shneiderman et al. explored dynamic queries in general and how these operations fit into a larger context of visual information seeking [87]. Ward introduced a visualization system based on multiple linked views with direct manipulation techniques including brushing and linking [105]. Anselin discussed how interactive visualization systems with linked views can be applied to Geographic

Information Systems [5]. 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 [112]. Techapichetvanich et al. explored how visualization interactions pertain to OLAP cubes in particular [96]. Sifer et al. introduced a visual interface utilizing coordinated dimension hierarchies for OLAP cubes [90]. Tegarden formulates some requirements for information visualization relevant for business applications, and highlights some unconventional interactive visualizations with potential application to data cube visualization [97].

1.1.4 Web Graphics Technology

The World Wide Web has evolved to become a full fledged application development platform [79]. HTML5 is the latest set of standards and APIs (Application Programming Interfaces) from the World Wide Web Consortium that define the capabilities of modern Web browsers [43]. HTML5 contains three graphics technologies that can support interactive Web-based visualizations: Canvas, SVG (Scalable Vector Graphics), and WebGL.

HTML5 Canvas provides a 2D immediate mode graphics API [35]. When using the Canvas API, developers must work with a stateful graphics context by issuing commands to manipulate the raster image of a Canvas element within the HTML page. This approach requires developers to manage rendering logic at a low level and manage data structures that correspond to graphical representations. The Canvas API has seen wide adoption for

HTML5-based games, however for visualization applications the higher level SVG API has seen wider adoption.

SVG (Scalable Vector Graphics) provides a 2D retained mode Graphics API [24]. SVG uses the HTML DOM (Document Object Model) to represent the definition of persistent graphical elements. When using SVG, developers need only be concerned with updating the DOM. The SVG engine within the browser is responsible for updating the display to correspond with the SVG DOM. In this way, SVG is a higher level API than Canvas. This makes SVG a preferred platform for developing visualizations. However, SVG is less optimizable than Canvas, because developers do not have access to the rendering logic. SVG has performance limitations relating to DOM manipulation overhead.

WebGL provides a 3D graphics API that is essentially a JavaScript interface to OpenGL ES [71]. OpenGL ES is a subset of OpenGL designed for use in embedded systems and mobile devices. Developers using WebGL must use programming techniques inherited from OpenGL such as buffer management, vertex management, shader definition, 3D projection, and lighting techniques. WebGL enables developers to take advantage of the GPU (Graphics Processing Unit) for massively parallel computation using shaders. WebGL supports high performance 2D and 3D graphics, but is much more complicated to use than Canvas or SVG. WebGL has been applied to interactive visualization of volumetric data [18].

Many high level libraries have been built for supporting use of Web graphics technologies. Three.js is a 3D scene graph library that includes rendering engines for all three graphics technologies [10]. Highcharts is a high level visualization library that provides pre-packaged chart types that can be customized to a limited extent. Leaflet is a library for creating tile-based geographic maps with zooming and panning. hBrowse is a generic framework introduced for Web-based hierarchy visualization [55]. Processing.js is a JavaScript port of the graphics language Processing using HTML5 Canvas. Many more libraries for Web-based graphics and visualization exist, but none have come close to the widespread adoption of D3.js.

D3.js is a flexible and powerful visualization library that uses SVG and has a strong community of users [9]. D3 at its core 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. Many supporting libraries for D3 have been created including Chart.js for composing visualization elements, Crossfilter.js for interactive multidimensional filtering, and DC.js for multiple linked views. Several reusable D3 chart libraries have appeared including NVD3, reD3, and dimple.

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 [102]. 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 [83]. GapMinder includes an animated scatter plot with an interactive time slider, a line chart showing statistics over time, and a world map. The Google Public Data Explorer provides a visual interface to selected public data sets similar to GapMinder, however it does not make the data available to users in a machine-readable format.

1.2 Pseudocode Conventions

Throughout this document, pseudocode is used to express data structures and algorithms. Our pseudocode is similar to that found in the book "Introduction to Algorithms" [19], 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 our pseudocode language.

1.2.1 Operators and Primitive Types

```
$x = 5$
```

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 pseucode 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.

1.2.2 Functions

```
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 anonymus 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.

1.2.3 Arrays

```
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  $myBooleanArray = [\text{TRUE}, \text{FALSE}, \text{TRUE}, \text{TRUE}]$
9  $myStringArray = [$'foo','bar'$]$
10 $numberOfBooleans = myBooleanArray.length$ // evaluates to 4
11 $numberOfStrings = myStringArray.length$ // evaluates to 2
12 for $str \in myStringArray$
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.

1.2.4 Objects

```
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 $values = keys.map(\lambda(property)$ **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.

1.2.5 The Event Loop

Our pseudocode assumes a single threaded execution environment with a built-in event loop, which may be implemented using the reactor pattern [84]. 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(λ() 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`.

TODO Discuss apply(fn, args)

TODO Discuss !bool

Chapter 2

The Universal Data Cube

A data cube is a multidimensional data structure that organizes data into *dimensions* and *measures*. Dimensions are sets of distinct entities that may be unordered, ordered, or hierarchical. Each distinct entity of a dimension is called a *member*. Measures are aggregated quantitative values that can be assigned to combinations of dimension members called *cells*. An *observation* links a cell with concrete values for one or more measures. A *data cube*, also referred to as simply a *cube*, is a set of observations that draw from a common set of dimensions and measures. The table below provides examples for each of the terms introduced.

Term	**Examples**
dimension	Space, Time
member	India, China, 1970, 2010
measure	Population, Gross Domestic Product
cell	India in 1970, China in 2010
observation	The population of India in 1970 was 555.2 million.
cube	The population of India in 1970 was 555.2 million.
	The population of India in 2010 was 1,206 million.
	The population of China in 1970 was 818.3 million.
	The population of China in 2010 was 1338 million.

Existing data cube technologies do not readily support integration of data cubes from multiple sources. Many data sets can be modeled as data cubes that have dimensions and measures in common. The dimensions and measures they have in common can be utilized to integrate multiple data cubes together into a unified structure. The difficulty in integrating data cubes from multiple sources arises from different identifiers referring to the same member and different scale factors being used for the same measure. Our data cube representation and integration framework, called the *Universal Data Cube* (UDC), addresses these difficulties.

In order to account for different identifiers referring to the same member, we introduce the concept of a *code list*. A code list is a family of identifiers, also called *codes*, that refer to members [23]. A *concordance table*, also referred to as simple a *concordance*, is a table that declares equivalences between codes from different code lists [28]. Members can be identified unambiguously as (dimension, code list, code) tuples called *qualified members*. The table below provides examples for each of the terms introduced.

Term	**Example**	
code list	Country Name, Country Code	
code	India, in	
qualified member	(Space, Country Code, in)	
	(Space, Country Name, China)	
concordance table	**Country Name**	**Country Code**
	India	in
	China	cn

Data tables can contain data that can be modeled as either a cube or a concordance. In either case, columns in the tables can be either *dimension columns* or *measure columns*. A dimension column contains codes from a single code list. A measure column contains numbers representing values for a certain measure. Values in a measure column may be scaled (multiplied by a scaling factor). In order to be imported into the UDC framework, data tables must be annotated with metadata that declares whether the table represents a cube or a concordance, and specifies the role of each column as either a dimension column or a measure column. Below are some concrete examples of data tables and their metadata.

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

```
dimensionColumns : [
    {column :'countryCode', dimension :'Space', codeList :'ISO3'},
    {column :'year', dimension :'Time', codeList :'Year'}
]
measureColumns : [
    {column :'gdp', measure :'Gross Domestic Product'}
]
```

countryName	unCountryCode	alphaCode
India	356	IND
China	156	CHN
United States of America	840	USA

```
dimensionColumns : [
    {column :'countryName', dimension :'Space', codeList :'UN Geoname'},
    {column :'unCountryCode', dimension :'Space', codeList :'UN M.49'},
    {column :'alphaCode', dimension :'Space', codeList :'ISO3'}
]
```

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

```
dimensionColumns : [
    {column :'countryCode', dimension :'Space', codeList :'UN M.49'},
    {column :'year', dimension :'Time', codeList :'Year'}
]
measureColumns : [
    {column :'pop', measure :'Population'}
]
```

Loading Concordances

Member Canonicalization

Loading Cubes

Cube Canonicalization

Merging Cubes

Querying Cubes

Cube Projection

Hierarchies

Drill Down and Roll Up

2.1 Data Sets

2.1.1 United Nations Population Estimates

2.1.2 World Bank World Development Indicators

2.1.3 United Nations Millenium Development Goals

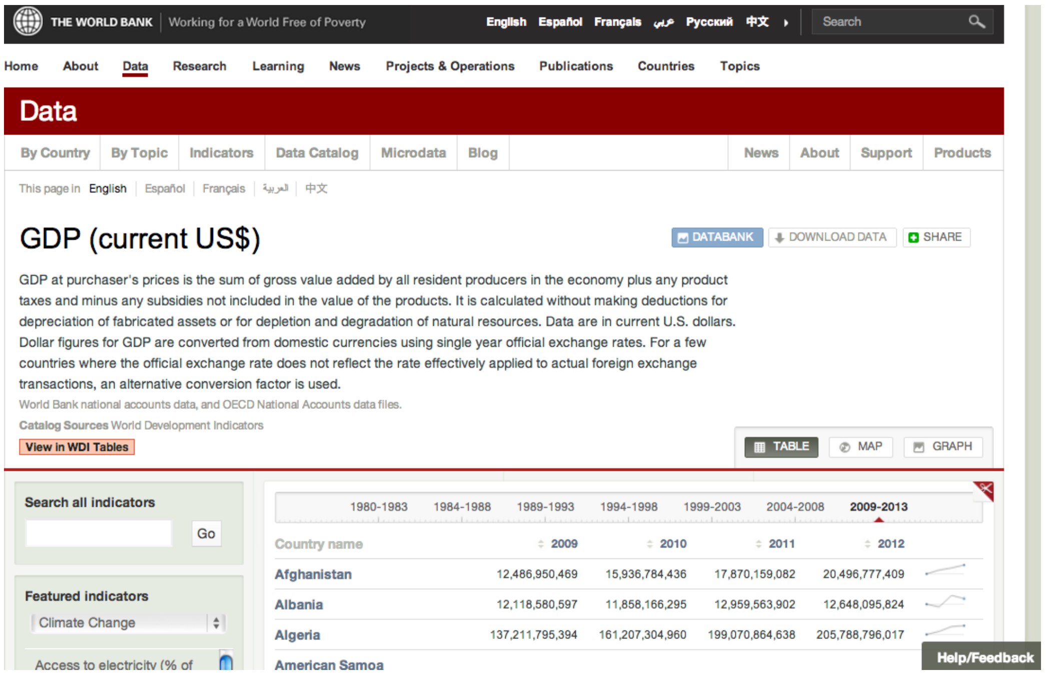
2.1.4 United States Census Population Estimates

TODO import data from http://www.census.gov/popest/data/state/totals/2013/index.html

2.1.5 US Central Intelligence Agency World Factbook

TODO import subsets of the data

Figure 2.1: The World Bank Web page for their data table containing Gross Domestic Product for world countries.


THE WORLD BANK | Working for a World Free of Poverty
English Español Français عربي Русский 中文
Search
Home About Data Research Learning News Projects & Operations Publications Countries Topics
Data
By Country By Topic Indicators Data Catalog Microdata Blog News About Support Products
This page in English Español Français العربية 中文
DATABANK DOWNLOAD DATA SHARE
View in WDI Tables
TABLE MAP GRAPH
Search all indicators
Go
Featured indicators
Climate Change
Access to electricity (% of
Help/Feedback

GDP (current US$)

GDP at purchaser's prices is the sum of gross value added by all resident producers in the economy plus any product taxes and minus any subsidies not included in the value of the products. It is calculated without making deductions for depreciation of fabricated assets or for depletion and degradation of natural resources. Data are in current U.S. dollars. Dollar figures for GDP are converted from domestic currencies using single year official exchange rates. For a few countries where the official exchange rate does not reflect the rate effectively applied to actual foreign exchange transactions, an alternative conversion factor is used.

World Bank national accounts data, and OECD National Accounts data files.

Catalog Sources World Development Indicators

1980-1983 1984-1988 1989-1993 1994-1998 1999-2003 2004-2008 2009-2013

Country name	2009	2010	2011	2012
Afghanistan	12,486,950,469	15,936,784,436	17,870,159,082	20,496,777,409
Albania	12,118,580,597	11,858,166,295	12,959,563,902	12,648,095,824
Algeria	137,211,795,394	161,207,304,960	199,070,864,638	205,788,796,017
American Samoa				

2.1.6 US Centers for Disease Control Causes of Death

TODO generalize stacked area and tree vis

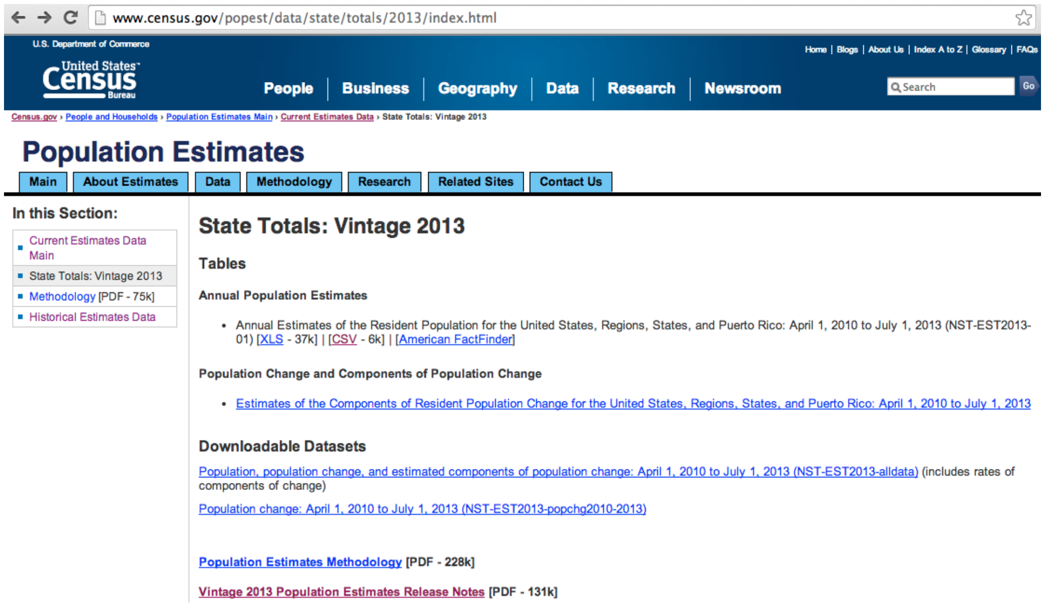
2.1.7 W3Schools Browser Market Share

TODO import this data completely

2.1.8 Natural Earth

TODO discuss data transformation process

Figure 2.2: A screenshot of the US Census Population by State data set. This data set is made available in Excel and CSV formats.



www.census.gov/popest/data/state/totals/2013/index.html

U.S. Department of Commerce

Home | Blogs | About Us | Index A to Z | Glossary | FAQs

United States Census Bureau

People | Business | Geography | Data | Research | Newsroom

Search Go

Census.gov › People and Households › Population Estimates Main › Current Estimates Data › State Totals: Vintage 2013

Population Estimates

Main | About Estimates | Data | Methodology | Research | Related Sites | Contact Us

In this Section:

- Current Estimates Data Main
- State Totals: Vintage 2013
- Methodology [PDF - 75k]
- Historical Estimates Data

State Totals: Vintage 2013

Tables

Annual Population Estimates

- Annual Estimates of the Resident Population for the United States, Regions, States, and Puerto Rico: April 1, 2010 to July 1, 2013 (NST-EST2013-01) [XLS - 37k] | [CSV - 6k] | [American FactFinder]

Population Change and Components of Population Change

- Estimates of the Components of Resident Population Change for the United States, Regions, States, and Puerto Rico: April 1, 2010 to July 1, 2013

Downloadable Datasets

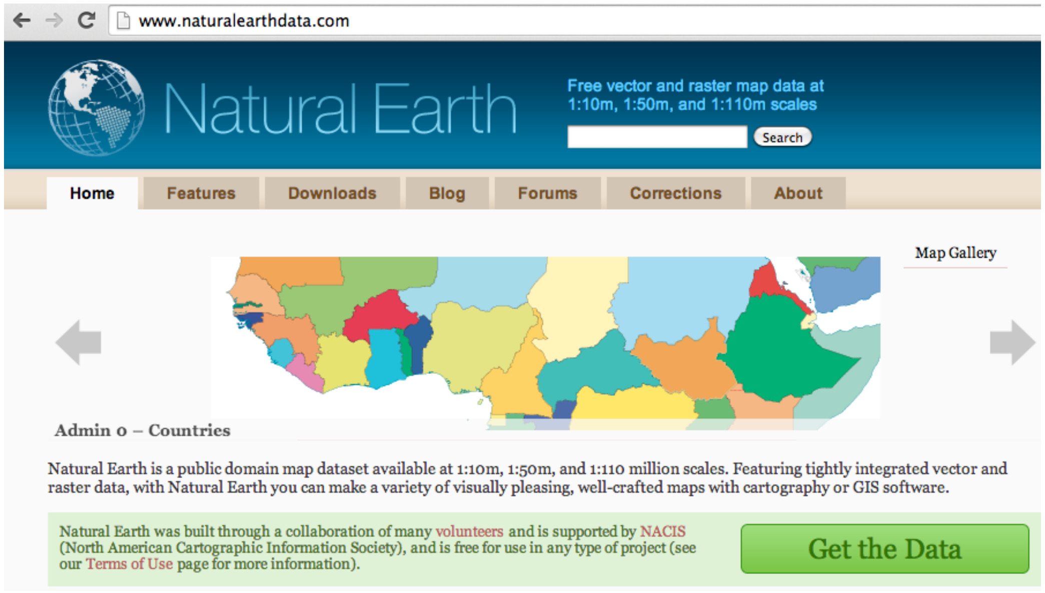
Population, population change, and estimated components of population change: April 1, 2010 to July 1, 2013 (NST-EST2013-alldata) (includes rates of components of change)

Population change: April 1, 2010 to July 1, 2013 (NST-EST2013-popchg2010-2013)

Population Estimates Methodology [PDF - 228k]

Vintage 2013 Population Estimates Release Notes [PDF - 131k]

Figure 2.3: A screenshot of the Natural Earth geographic data Web site.



www.naturalearthdata.com

Natural Earth

Free vector and raster map data at 1:10m, 1:50m, and 1:110m scales

Search

Home | Features | Downloads | Blog | Forums | Corrections | About

Map Gallery

Admin 0 – Countries

Natural Earth is a public domain map dataset available at 1:10m, 1:50m, and 1:110 million scales. Featuring tightly integrated vector and raster data, with Natural Earth you can make a variety of visually pleasing, well-crafted maps with cartography or GIS software.

Natural Earth was built through a collaboration of many volunteers and is supported by NACIS (North American Cartographic Information Society), and is free for use in any type of project (see our Terms of Use page for more information).

Get the Data

Chapter 3

Functional Reactive Models

Functional reactive programming allows developers to declaratively specify data dependency graphs [104]. Functional reactive programming has been applied to interactive graphics [32] and robotics [46]. The Model View Controller paradigm is an approach to cleanly separate application operations into three classes. The *model* contains the data structures representing the application state. The *view* handles graphical presentation of the model to the user. The *controller* translates user interactions (such as mouse clicks, key presses, or multi-touch gestures) into changes in the model [57]. The Model View Controller paradigm can be combined with functional reactive programming to enable straightworward creation of reactive systems based on data flow graphs.

3.1 The Model Data Structure

The Model in the Model View Controller (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 [59]. This kind of model can fulfill the all of 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.

We first discuss a simple model with only *set* and *get* methods ($SimplestModel$), then discuss a more complex version that also includes *on* ($SimpleModel$), then finally introduce a reactive model that includes the *when* operator from functional reactive programming ($Model$).

3.1.1 Simplest Model

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

The above 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 called *values* that will contain the key-value mapping. Lines 3 - 5 define the *set* and *get* methods, which store and retreive values from the internal *values* object. Here's an example of how *SimplestModel* might be used.

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

3.1.2 Simple Model

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

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

3.1.3 Functional Reactive Model

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 [1]. A data flow graph defines a directed acyclic graph of data dependencies. The data flow model is amenable to construction of visual programming languages [44]. 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 [32].

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 declaratively specify data dependencies, and change propagation should be automatically managed. 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 and

RXJS).

Our Model 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 = λ(callback)
2      queued = FALSE
3      return λ()
4          if queued == FALSE
5              queued = TRUE
6              run(λ()
7                  queued = FALSE
8                  callback()
9              )
```

The *debounce*(*callback*) function returns a function that, when invoked one or more times in a single codepath, 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 (using the built-in function *run* discussed in section 1.2.5).

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.

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

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 (discussed in section 1.2.4). 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 the function returns TRUE (on line 5). We are now ready to define our model that includes the *when* operator.

```
$Model = \lambda()$
    $simpleModel = SimpleModel()$
    **return**
        $set : simpleModel.set$
        $get : simpleModel.get$
        $when : \lambda(dependencies, fn)$
            $callFn = debounce(\lambda()$
                $args = dependencies.map(simpleModel.get)$
                **if** $allAreDefined(args)$
                    $apply(fn, args)$
            $)$
            $callFn()$
            **for** $key \in dependencies$
                $simpleModel.on(key, callFn)$
```

TODO describe Model() **TODO** add pseudocode for each figure

Figure 3.1: A simple data dependency graph using functional reactive models. Here, $fullName$ is recomputed whenever $firstName$ or $lastName$ change.

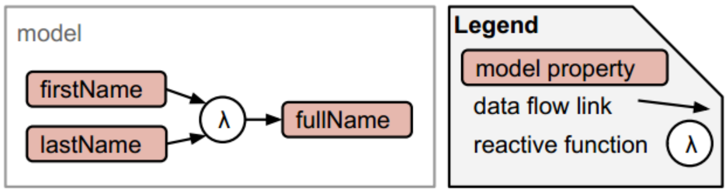
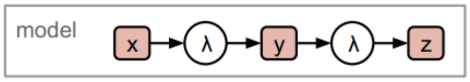


Figure 3.2: A data dependency graph with two hops. When x changes, the change propagates to y then to z.



3.2 Functional Reactive Visualizations

Figure 3.3: The data dependency graph for a dynamic bar chart.

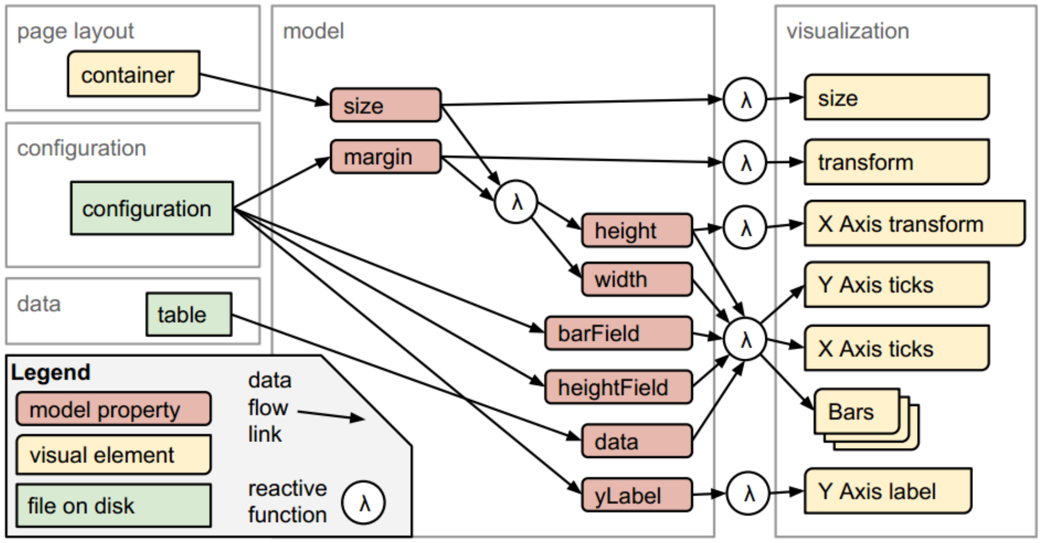


Figure 3.4: The appearance of a dynamic bar chart.

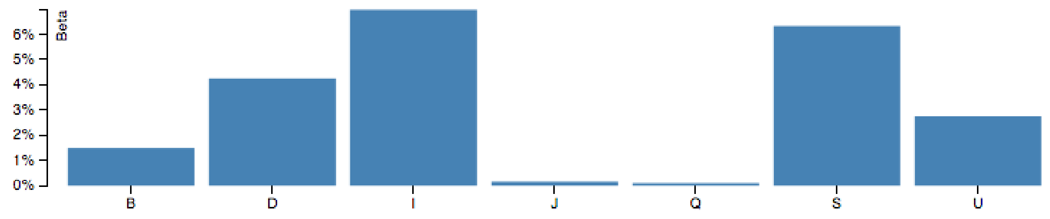
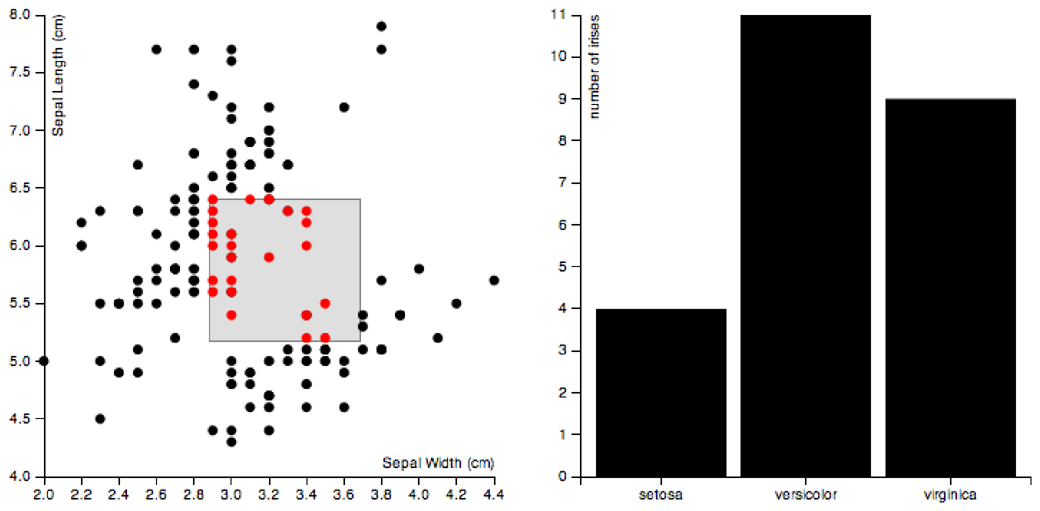


Figure 3.5: An example of multiple linked views using functional reactive models. Brushing in the scatter plot causes the selected data to be aggregated and plotted in the bar chart. Data shown is Fisher's Iris data [34].



Chapter 4

Visualizations

4.1 Data Cubes and Visualization Theory

Data cubes can contain the following kinds of data:

- *nominal* - unordered collections of categories
- *ordinal* - ordered collections of categories
- *hierarchical* - collections of categories organized as trees
- *quantitative* - continuously varying numeric values

Data cube dimensions can be nominal, ordinal or hierarchical. Data cube measures are always quantitative. This mapping relates data cubes to data types that have been well studied in the literature on visualization theory [7, 64, 36]. **TODO** include pseudocode for each visualization

4.2 Bar Chart

4.3 Scatter Plot

4.4 Line Chart

4.5 Choropleth Map

4.6 Stacked Area Plot

4.7 TreeMap

4.8 Node Link Tree

4.9 Radial Tree

4.10 Icicle Plot

4.11 TreeMap

4.12 Parallel Coordinates

Chapter 5

Visualization Dashboard Infrastructure

5.1 Dashboard Layout using Nested Boxes

TODO Include nested box layout pseudocode

5.2 Multiple Linked Views

TODO Include generic linking pseudocode using ”when”

5.3 Dynamic Dashboard Configuration

TODO Include pseudocode for computing configuration diffs

Chapter 6

UDC Visualization Dashboards

Chapter 7

Collaboration and History Navigation

Bibliography

[1] Greg Abram and Lloyd Treinish. An extended data-flow architecture for data analysis and visualization. In *Proceedings of the 6th conference on Visualization'95*, page 263. IEEE Computer Society, 1995.

[2] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. Modeling multidimensional databases. In *Data Engineering, 1997. Proceedings. 13th International Conference on*, pages 232–243. IEEE, 1997.

[3] Christopher Ahlberg. Spotfire: an information exploration environment. *ACM SIGMOD Record*, 25(4):25–29, 1996.

[4] Akiko Aizawa and Keizo Oyama. 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*, pages 30–39. IEEE, 2005.

[5] Luc Anselin. Interactive techniques and exploratory spatial data analysis. *Geographical Information Systems: principles, techniques, management and applications*, 1:251–264, 1999.

[6] Richard A Becker and William S Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.

[7] Jacques Bertin. Semiology of graphics: diagrams, networks, maps. 1983.

[8] Markus Blaschka, Carsten Sapia, Gabriele Hofling, and Barbara Dinter. Finding your way through multidimensional data models. In *Database and Expert Systems Applications, 1998. Proceedings. Ninth International Workshop on*, pages 198–203. IEEE, 1998.

[9] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[10] Ricardo Cabello. Three. js. *URL: https://github. com/mrdoob/three. js*, 2013.

[11] Stuart K Card and Jock Mackinlay. The structure of the information visualization design space. In *Information Visualization, 1997. Proceedings., IEEE Symposium on*, pages 92–99. IEEE, 1997.

[12] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[13] Ed Huai-hsin Chi. A taxonomy of visualization techniques using the data state reference model. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pages 69–75. IEEE, 2000.

[14] Ed Huai-hsin Chi and John T Riedl. An operator interaction framework for visualization systems. In *Information Visualization, 1998. Proceedings. IEEE Symposium on*, pages 63–70. IEEE, 1998.

[15] Peter Christen. 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*, pages 1065–1068. ACM, 2008.

[16] James Clifford and Abdullah Uz Tansel. On an algebra for historical relational databases: two views. In *ACM SIGMOD Record*, volume 14, pages 247–265. ACM, 1985.

[17] Edgar F Codd, Sharon B Codd, and Clynch T Salley. Providing olap (on-line analytical processing) to user-analysts: An it mandate. *Codd and Date*, 32, 1993.

[18] John Congote, Alvaro Segura, Luis Kabongo, Aitor Moreno, Jorge Posada, and Oscar Ruiz. Interactive visualization of volumetric data with webgl in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology*, pages 137–146. ACM, 2011.

[19] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms.* MIT press, 2009.

[20] Alfredo Cuzzocrea and Svetlana Mansmann. Olap visualization: models, issues, and techniques. *Encyclopedia of Data Warehousing and Mining,*, pages 1439–1446, 2009.

[21] Alfredo Cuzzocrea, Domenico Saccà, and Paolo Serafino. A hierarchy-driven compression technique for advanced olap visualization of multidimensional data cubes. In *Data Warehousing and Knowledge Discovery*, pages 106–119. Springer, 2006.

[22] Alfredo Cuzzocrea, Domenico Sacca, and Paolo Serafino. Semantics-aware advanced olap visualization of multidimensional data cubes. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(4):1–30, 2007.

[23] Richard Cyganiak, Dave Reynolds, and Jeni Tennison. The rdf data cube vocabulary. Technical Report http://www.w3.org/TR/vocab-data-cube, W3C, December 2013.

[24] Erik Dahlström, Patrick Dengler, Anthony Grasso, Chris Lilley, Cameron McCormack, Doug Schepers, Jonathan Watt, Jon Ferraiolo, Fujisawa Jun, and Dean Jackson. Scalable vector graphics (svg) 1.1 (second edition). *W3C Recommendation*, 2011.

[25] Anindya Datta and Helen Thomas. The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses. *Decision Support Systems*, 27(3):289–301, 1999.

[26] AnHai Doan, Pedro Domingos, and Alon Y Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM Sigmod Record*, volume 30, pages 509–520. ACM, 2001.

[27] AnHai Doan, Pedro Domingos, and Alon Y Levy. Learning source description for data integration. In *WebDB (Informal Proceedings)*, pages 81–86, 2000.

[28] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration.* Elsevier, 2012.

[29] AnHai Doan and Alon Y Halevy. Semantic integration research in the database community: A brief survey. *AI magazine*, 26(1):83, 2005.

[30] Stephen G Eick. Visualizing multi-dimensional data. *ACM SIGGRAPH computer graphics*, 34(1):61–67, 2000.

[31] Mohamed G Elfeky, Vassilios S Verykios, and Ahmed K Elmagarmid. Tailor: A record linkage toolbox. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 17–28. IEEE, 2002.

[32] Conal Elliott and Paul Hudak. Functional reactive animation. In *ACM SIGPLAN Notices*, volume 32, pages 263–273. ACM, 1997.

[33] Ronald Fagin, Phokion G Kolaitis, Renée J Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Database Theory—ICDT 2003*, pages 207–224. Springer, 2003.

[34] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[35] Steve Fulton and Jeff Fulton. *HTML5 Canvas*. O'Reilly Media, 2013.

[36] Martin Graham and Jessie Kennedy. A survey of multiple tree visualisation. *Information Visualization*, 9(4):235–252, 2010.

[37] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[38] Lifang Gu, Rohan Baxter, Deanne Vickers, and Chris Rainsford. Record linkage: Current practice and future directions. *CSIRO Mathematical and Information Sciences Technical Report*, 3:83, 2003.

[39] Marc Gyssens and Laks VS Lakshmanan. A foundation for multi-dimensional databases. In *VLDB*, volume 97, pages 106–115, 1997.

[40] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: the teenage years. In *Proceedings of the 32nd international conference on Very large data bases*, pages 9–16. VLDB Endowment, 2006.

[41] Pat Hanrahan, Chris Stolte, and Jock Mackinlay. visual analysis for everyone. *Tableau White paper*, 4, 2007.

[42] Bin He and Kevin Chen-Chuan Chang. Statistical schema matching across web query interfaces. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 217–228. ACM, 2003.

[43] Ian Hickson, Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, and Silvia Pfeiffer. Html5: A vocabulary and associated apis for html and xhtml. *W3C Working Draft edition*, 2013.

[44] Daniel D Hils. Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing*, 3(1):69–101, 1992.

[45] CD'Arcy J Holman, A John Bass, Ian L Rouse, and Michael ST Hobbs. 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):453–459, 1999.

[46] Paul Hudak, Antony Courtney, Henrik Nilsson, and John Peterson. Arrows, robots, and functional reactive programming. In *Advanced Functional Programming*, pages 159–187. Springer, 2003.

[47] Matthew A Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[48] Matthew A Jaro. Probabilistic linkage of large public health data files. *Statistics in medicine*, 14(5-7):491–498, 1995.

[49] Jérôme, Pavel Shvaiko, et al. *Ontology matching*. Springer, 2007.

[50] Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *The knowledge engineering review*, 18(1):1–31, 2003.

[51] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *PART 5———Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 3363–3372. ACM, 2011.

[52] Jaewoo Kang and Jeffrey F Naughton. On schema matching with opaque column names and data values. In *SIGMOD Conference*, pages 205–216, 2003.

[53] Daniel A Keim. Information visualization and visual data mining. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):1–8, 2002.

[54] Ralph Kimball. *The data warehouse lifecycle toolkit: expert methods for designing, developing, and deploying data warehouses.* Wiley. com, 1998.

[55] Lukasz Kokoszkiewicz, Julia ANDREEVA, Ivan Antoniev DZHUNOV, Edward KARAVAKIS, Massimo LAMANNA, Jakub MOSCICKI, and Laura SARGSYAN. hbrowse-generic framework for hierarchical data visualization. In *Proceedings of the Computing in High Energy and Nuclear Physics 2012 International Conference*, 2012.

[56] Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. Record linkage: similarity measures and algorithms. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 802–803. ACM, 2006.

[57] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.

[58] Hing-Yan Lee and Hwee-Leng Ong. A new visualisation technique for knowledge discovery in olap. In *KDOOD/TDOOD*, pages 23–25. Citeseer, 1995.

[59] Avraham Leff and James T Rayfield. Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, pages 118–127. IEEE, 2001.

[60] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[61] Chang Li and X Sean Wang. A data model for supporting on-line analytical processing. In *Proceedings of the fifth international conference on Information and knowledge management*, pages 81–88. ACM, 1996.

[62] Lauro Lins, James T Klosowski, and Carlos Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2456–2465, 2013.

[63] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. immens: Real-time visual querying of big data. *Eurovis*, 2013.

[64] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics (TOG)*, 5(2):110–141, 1986.

[65] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *VLDB*, volume 1, pages 49–58, 2001.

[66] Jayant Madhavan, S Jeffery, Shirley Cohen, X Dong, David Ko, Cong Yu, and Alon Halevy. Web-scale data integration: You can only afford to pay as you go. In *Proceedings of CIDR*, pages 342–350, 2007.

[67] Andreas S Maniatis, Panos Vassiliadis, Spiros Skiadopoulos, and Yannis Vassiliou. Advanced visualization for olap. In *Proceedings of the 6th ACM international workshop on Data warehousing and OLAP*, pages 9–16. ACM, 2003.

[68] Svetlana Mansmann and Marc H Scholl. *Extending visual OLAP for handling irregular dimensional hierarchies*. Springer, 2006.

[69] Svetlana Mansmann and Marc H Scholl. Exploring olap aggregates with hierarchical visualization techniques. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 1067–1073. ACM, 2007.

[70] Svetlana Mansmann and Marc H Scholl. Visual olap: A new paradigm for exploring multidimensional aggregates. In *Proc. of IADIS Int'l Conf. on Computer Graphics and Visualization (CGV)*, pages 59–66, 2008.

[71] Kouichi Matsuda and Rodger Lea. *WebGL programming guide: interactive 3D graphics programming with WebGL*. Pearson Education, 2013.

[72] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *VLDB*, volume 98, pages 24–27. Citeseer, 1998.

[73] United Nations. World population prospects: The 2012 revision. `http://esa.un.org/unpd/wpp/Excel-Data/population.htm`, February 2014.

[74] Chris North, Nathan Conklin, Kiran Indukuri, and Varun Saini. Visualization schemas and a web-based architecture for custom multiple-view visualization of multiple-table databases. *Information Visualization*, 1(3-4):211–228, 2002.

[75] Chris North and Ben Shneiderman. Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In *Proceedings of the working conference on Advanced visual interfaces*, pages 128–135. ACM, 2000.

[76] Natalya F Noy. Semantic integration: a survey of ontology-based approaches. *ACM Sigmod Record*, 33(4):65–70, 2004.

[77] Natalya F Noy. Ontology mapping. In *Handbook on ontologies*, pages 573–590. Springer, 2009.

[78] Natalya F Noy and Mark A Musen. The prompt suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.

[79] Mark Pilgrim. *HTML5: up and running.* " O'Reilly Media, Inc.", 2010.

[80] William Playfair. Commercial and political atlas: Representing, by copper-plate charts, the progress of the commerce, revenues, expenditure, and debts of england, during the whole of the eighteenth century. *London: Corry*, 1786.

[81] Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001.

[82] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems.* Osborne/McGraw-Hill, 2000.

[83] Hans Rosling, Rönnlund A Rosling, and Ola Rosling. New software brings statistics beyond the eye. *Statistics, Knowledge and Policy: Key Indicators to Inform Decision Making. Paris, France: OECD Publishing*, pages 522–530, 2005.

[84] Douglas C Schmidt. Reactor: An object behavioral pattern for concurrent event demultiplexing and dispatching. 1995.

[85] Mathew Scotch, Bambang Parmanto, and Valerie Monaco. Usability evaluation of the spatial olap visualization and analysis tool (sovat). *Journal of Usability Studies*, 2(2):76–95, 2007.

[86] Matthew Scotch and Bambang Parmanto. Sovat: Spatial olap visualization and analysis tool. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 142b–142b. IEEE, 2005.

[87] Ben Shneiderman. Dynamic queries for visual information seeking. *Software, IEEE*, 11(6):70–77, 1994.

[88] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[89] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. In *Journal on Data Semantics IV*, pages 146–171. Springer, 2005.

[90] Mark Sifer. A visual interface technique for exploring olap data with coordinated dimension hierarchies. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 532–535. ACM, 2003.

[91] Chris Stolte, Diane Tang, and Pat Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):52–65, 2002.

[92] Chris Stolte, Diane Tang, and Pat Hanrahan. Query, analysis, and visualization of hierarchically structured data using polaris. In *KDD*, volume 2, pages 112–122. Citeseer, 2002.

[93] Chris Stolte, Diane Tang, and Pat Hanrahan. Multiscale visualization using data cubes. *Visualization and Computer Graphics, IEEE Transactions on*, 9(2):176–187, 2003.

[94] Deborah F Swayne, Dianne Cook, and Andreas Buja. Xgobi: Interactive dynamic data visualization in the x window system. *Journal of Computational and Graphical Statistics*, 7(1):113–130, 1998.

[95] Deborah F Swayne, Duncan Temple Lang, Andreas Buja, and Dianne Cook. Ggobi: evolving from xgobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis*, 43(4):423–444, 2003.

[96] Kesaraporn Techapichetvanich and Amitava Datta. Interactive visualization for olap. In *Computational Science and Its Applications–ICCSA 2005*, pages 206–214. Springer, 2005.

[97] David P Tegarden. Business information visualization. *Communications of the AIS*, 1(1es):4, 1999.

[98] Edward R Tufte and PR Graves-Morris. *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT, 1983.

[99] Michael Uschold and Michael Gruninger. Ontologies and semantics for seamless connectivity. *ACM SIGMod Record*, 33(4):58–64, 2004.

[100] Panos Vassiliadis. Modeling multidimensional databases, cubes and cube operations. In *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on*, pages 53–62. IEEE, 1998.

[101] Panos Vassiliadis and Timos Sellis. A survey of logical models for olap databases. *ACM Sigmod Record*, 28(4):64–69, 1999.

[102] Fernanda B Viegas, Martin Wattenberg, Frank Van Ham, Jesse Kriss, and Matt McKeon. Manyeyes: a site for visualization at internet scale. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1121–1128, 2007.

[103] Holger Wache, Thomas Voegele, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann, and Sebastian Hübner. Ontology-based integration of information-a survey of existing approaches. In *IJCAI-01 workshop: ontologies and information sharing*, volume 2001, pages 108–117. Citeseer, 2001.

[104] Zhanyong Wan and Paul Hudak. Functional reactive programming from first principles. In *ACM SIGPLAN Notices*, volume 35, pages 242–252. ACM, 2000.

[105] Matthew O Ward. Xmdvtool: Integrating multiple methods for visualizing multivariate data. In *Proceedings of the Conference on Visualization'94*, pages 326–333. IEEE Computer Society Press, 1994.

[106] Leland Wilkinson. *The grammar of graphics*. Springer, 2005.

[107] William E Winkler. Matching and record linkage. *Business survey methods*, 1:355–384, 1995.

[108] William E Winkler. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*. Citeseer, 1999.

[109] William E Winkler. Overview of record linkage and current research directions. In *Bureau of the Census*. Citeseer, 2006.

[110] Jing Yang, Matthew O Ward, and Elke A Rundensteiner. Interactive hierarchical displays: a general framework for visualization and exploration of large multivariate data sets. *Computers & Graphics*, 27(2):265–283, 2003.

[111] Qiang Yang, Joshua Zhexue Huang, and Michael Ng. A data cube model for prediction-based web prefetching. *Journal of Intelligent Information Systems*, 20(1):11–30, 2003.

[112] Ji Soo Yi, Youn ah Kang, John T Stasko, and Julie A Jacko. Toward a deeper understanding of the role of interaction in information visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1224–1231, 2007.

[113] Patrick Ziegler and Klaus R Dittrich. Three decades of data integration-all problems solved? In *IFIP congress topical sessions*, pages 3–12. Springer, 2004.