Master's Thesis

Multiple Coordinated Views on Massive Geo Data

Multiple koordinierte Visualisierungen großer Mengen an Geodaten

Robert Schäfer

robert.schaefer@student.hpi.uni-potsdam.de

Hasso Plattner Institute for Digital Engineering
Department of Computer Science

Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
https://hpi.de/doellner/

Supervisors:

Prof. Dr. Jürgen Döllner
Benjamin Hagedorn
Jan Klimke

Hasso Plattner Institute
Potsdam, Germany
November 21, 2017

Numerous visualization techniques exist for data-driven decision support systems. Those systems primarily used to give guidance in geopolitical decisions often deal with hierarchical and geographical data. There are well-researched data visualizations for both kinds individually. For geographical there are choropleth maps, flow maps, bubble maps and for hierarchical data there are treemaps. To this day however, no appropriate data visualization for hierarchical and geographical information in one view exists. In this thesis, we evaluate coordinated multiple views to combine the capabilities of both data visualizations. We describe a framework to coordinate arbitrary data visualizations in theoretical terms and we give a reference implementation.

Contents

List of Figures v

List of Tables vii

1 Introduction 1

1.1 Motivation 1

1.2 Problem Statement 2

1.3 Hypothesis 2

1.4 Contributions 3

1.5 Structure of the Work 3

2 Related Work 5

2.1 Foundations 5

2.1.1 Data Visualizations 5

2.1.2 Visual Variables 6

2.1.3 Treemaps 6

2.1.4 Geographical Data Visualization 8

2.1.5 Coordinated Multiple Views 11

2.2 Interaction Theory 12

2.2.1 Interaction Categories 12

2.2.2 Interaction Models 13

3 Analysis 15

3.1 Single Visualization Interactions 15

3.2 Multiple View Interactions 22

3.3 Requirements 22

4 Concept 25

4.1 Terminology 25

4.2 Formalization of an interaction 27

4.3 Conceptual Framework 27

4.4 Shared data model 28
4.5 Predefined Interaction Functions 29
4.6 Communication pattern 32

5 Implementation 35
5.1 Architecture 35
5.2 Software patterns 37
5.3 Geographical visualization with React and Leaflet 39
5.3.1 GeoJSON component 40
5.4 MultiviewController 42
5.5 Implemented interactions 44

6 Evaluation 47
6.1 Comparison of client-side component frameworks 47
6.2 Use case 49
6.2.1 Data Sets 49
RISO 50
Rundfunk MITBESTIMMEN 51
6.3 Existing Interactions 53
6.4 Planned Interactions 55

7 Conclusion and Future Work 57

List of Figures

2.1 Bertin's [5] original visual variables. 7
2.2 First hierarchy level of German exports 7
2.3 Second hierarchy level of German exports 8
2.4 Example of a 2.5D tree map 9
2.5 Charles Minard's map of Napoleon's disastrous Russian campaign of 1812. Minard managed to represent six values in two graphical dimensions: The number of Napoleon's troops, the travelled direction and distance, latitude and longitude relative to specific dates and the temperature. 9
2.6 Choropleth Map of Obesity in the United States in 2008. Present of population classified as "obese" (Body Mass Index in excess of 30), by state. 10
2.7 Airline on-time performance: Correlation of time of day with arrival delay. Most recent flight with a delay of more than 100 minutes selected. 11

3.1 Line charts are used to display trends 15
3.2 A multiset bar charts is a variation of a bar chart 16
3.3 A histogram is a bar chart over a continuous interval 17
3.4 Bubble charts and scatter plots are similar regarding interactions 18
3.5 Stacked bar charts can be ordered along a baseline or stretch to 100% width to show the percentage-of-the-whole of each group . . 19
3.6 A population pyramid can be modeled as a stacked bar chart . . . 19
3.7 Tree maps and sunburst diagrams are ideal to show hierarchies . . 20
3.8 Choropleth maps focus on a density while flow maps show a migration of data 21
3.9 Similar to a calendar, a gantt chart shows activities and the progress along a time line 21

3.10 The screenshot-tool “shutter” shows a magnified detail view of the area around the mouse cursor in the lower right corner of the screen . . . 22

4.1 A data structure for tabular, hierarchical and relational data . . . 28
4.2 The message flow of the communication pattern: Every view can decide for itself whether to trigger an interaction or respond to an interaction with a visual effect. . . . 32

5.1 Architecture of the reference implementation. . . . 36
5.2 The sequence diagram shows the notification of different components. The user first chooses a feature set and hovers over a polygon in the geographical map. . . . 37
5.3 Implementation of the component pattern in EmberJS. The example shows a page of a webshop. The customer is about to order the items in the shopping cart. . . . 38
5.4 User distribution of Rundfunk MITBESTIMMEN across German federal states . . . 45

6.1 Comparison of frontend frameworks . . . 49
6.2 Part of the `RISO` database schema. Primary keys are set in bold. 50
6.3 Database schema of the Rundfunk MITBESTIMMEN application . . . 52
6.4 Items can be highlighted with a click, Bavaria is currently highlighted. A mouse over reveals a tooltip showing item properties. The menu on the left side allows to change the data set and the specific base visualization. . . . 54
6.5 Only gas stations with a price for E10 within 1.349 Euro and 1.589 Euro are displayed in the map . . . 54

List of Tables

3.1 Interactions for line charts 16
3.2 Interactions for bar charts 17
3.3 Interactions for bubble charts 18
3.4 Interactions for stacked bar charts 19
3.5 Interactions for hierarchical visualizations 20
3.6 Interactions for geographical visualizations 21
3.7 Interactions for temporal visualizations 22

1 Introduction

The human brain processes visual information better than it processes text. As a result, the most tangible data analyses usually come with some sort of data visualization. On a computer, the user can interact with the data and explore different levels of granularity. The visualization changes and the user can iteratively perform another interaction. In many cases a great interactivity results in a great user experience.

There is a wide range of existing frameworks and implementations for data visualizations. In some advanced cases, the data is visualized in multiple views which are linked with each other. These coordinated multiple views are interesting, as they try to make the most out of the various advantages of different visualizations. However, implementing interactions on data visualizations, and especially on multiple visiualization, is tedious and costly. There is currently no extensible framework that helps to implement interactions of the same data in various visualizations.

Establish the niche, why is there further research on your topic?

Introduce the current research, what's the hypothesis, the research question?

1.1 Motivation

We create data visualizations of multi-dimensional, hierarchical and geographical data. Namely, we develop Rundfunk MITBESTIMMEN which is an application for German citizen to publish which public broadcasts should benefit from their broadcasting fees. The output of this application is a public user ranking and it can be used by broadcasting corporations to evaluate their program.

Visualizations of multidimensional, hierarhical data guide media researchers, journalists and the general audience to draw conclusions.

To explain this a little deeper: Public broadcasting in Germany is organized federally, a German home belongs to the jurisdiction of a public broadcasting corporation. But the produced content can be used by all German citizen and it is even required to be free and available for everyone. In the application, common users vote on the entire collection of broadcasts but media researchers are only interested in users of a service area. User accounts have a local context and broadcasts have a global context.

Let's say a media researcher is interacting with two views of the data. The interaction may happen in one view but is reflected in another view. E.g. the researcher selects user accounts from a geographical map but has the intention to update a separate view. That view could show e.g. a listing of the most popular broadcasts, based on the data of user accounts from that area.

Having the interaction spread across several views creates a great value for the user but the implementation this functionality is very tedious. We see a strong demand for coordinated multiple views and the development effort as the main obstacle. A coordinated multiple view composed of arbitrary charts and plots and the implementation of their interactions turns into an unsustainable amount of work.

1.2 Problem Statement

A 2.5D tree map of hierarchical, geographical data can loose the geographic context if the tiling algorithm is based on non-geographical features. Items that should belong together according to their geographic circumstances may be scattered across the 2.5D tree map. This impairs the comprehensibility and complicates the selection of geographical units of items.

1.3 Hypothesis

A second, geographical visualization next to the 2.5D tree map can maintain the geographical context. If an interaction in one visualization is reflected in the other, this further supports the analysis of the data. Moreover, the limitations of a single data visualization can be avoided by interacting with the data trough another view.

1.4 Contributions

In order to validate the conceptual framework for coordinated multiple views, we develop a reference implementation to investigate its feasibility. The development of the reference implementation is subdivided into the following packages:

1. Concept

   Based on a number of examples, a basic, conceptual framework is designed. This includes a common data structure, a formalization of an interaction in general and the communication protototocol for multiple views.

2. Implementation

   We develop a geographical represenatation of the data. Items in the map can be focused, highlighted, selected with multiple clicks or with a bounding box. This creates a powerful selection mechanism, which can be used to select data in map-based representations and highlight the data in the 2.5D tree map.

3. Integration of 2.5D tree map and 2D map We develop linking and interaction mechanisms between 2D maps (for map-based representation) and 2.5D tree maps (for abstract information representation).

4. Demonstration and evaluation

   Coordinated multiple view layouts and suitable views are implemented and tested for the selected test data. Based on the test data sets, the coordinated multiple view implementation is examined and evaluated for design criteria [42], general usability aspects [37] and usage for typical visual analytics tasks.

1.5 Structure of the Work

In Section 2 we introduce basic terminology of coordinated multiple views and the theoretical background in this area of research. This Section also covers the state of the art and research on multiple views and coordinated interactions. In Section 3 we analyze a set of data visualizations and their interactions by example. The gained knowledge from that section is used in the following Section 4 for a formalization of the coordinated multiple view framework. In Section 5 we describe the reference implementation. Then in Section 6 we take the requirements from Section 3 to evaluate the conceptual framework and its implementation. Finally sum up the main contribution in Section 7 and outline the future work.

2

Related Work

no section without text

2.1 Foundations

no section without text

2.1.1 Data Visualizations

Data visualizations is a means of visual communication and have steadily developed since the 16th century [18]. Otherwise abstract information is visually represented, making complex data more accessible, understandable and usable. Kusinitz [28] explains that the human brain processes visual information 60,000 times faster than text and visual content makes up even 93% of all human communication. The purpose of data visualizations is twofold, according to the Interaction Design Foundation: sense-making and communication. Statistical information is abstract and in data visualization “we must find a way to give form to that which has none.” [16] Successful data visualizations helps the human user to derive knowledge and meta data from the visualization itself, Nocke and Schumann [34] call it “visual data mining”.

Data-driven decision support systems are applications to support businesses and organizational decision-making activities in which data visualizations play a key part [29] [35]. Visualizations are an obvious choice for managers who demand a quick overview on performance data. Stephen Few’s book “Show me the numbers” was named after the phrase often used by sales managers who can’t afford to wade through lengthy reports.

Prof. Döllner mentioned the emerging “cockpit” views

We can expect to see these technologies more in more in business applications. McAfee and Brynjolfsson [33] from the MIT Center of Digital Business showed that organizations driven most by data-based decision making had 4% higher productivity rates and 6% higher profits. However, little research has been done regarding the performance of coordinated multiple views in the field of decision making. There might be a great potential. Back in 1997 Mayer [32] conducted eight studies to compare the effect of using multimedia on university students. The studies showed that when using combined visual and verbal explanations the generation of creative problem solutions increased by an average of more than 50%.

So the application of combined data visualization techniques in decision making seems to be a promising strategy. Nevertheless is is unclear, which visualization techniques are the most suitable to be used in combination. If we know what kind of data we are dealing with, what are the best suited visualization techniques? Let’s say we have multidimensional data, is there an order in how people access these multiple dimensions? How do these visualizations perform and what are best practices to be considered for their implementation?

2.1.2 Visual Variables

French cartographer Jaques Bertin introduced seven visual varables in 1967 [5]. We can see an example for each in Figure 2.1. These visual variables are used in cartography but can also be applied to data visualization in general. Carpendale [11] explains in detail their use in terms of computational information instead of printed cartography. Garlandini and Fabrikant [19] put these visual variables under systematical validation procedures. The authors conclude that the variable size provides the most accurate and efficent performance while the variable orientation provides the least performance.

2.1.3 Treemaps

The visualization of hierarchical data has a long tradition. The traditional visual representation of a tree is a directed graph with the root node at the top. An everyday use case is a directory tree example of a file system, e.g. in file browsers or command line utilities like `tree` in UNIX based operating systems. As Shneiderman [40] mentions, this visualization becomes increasingly large when displaying

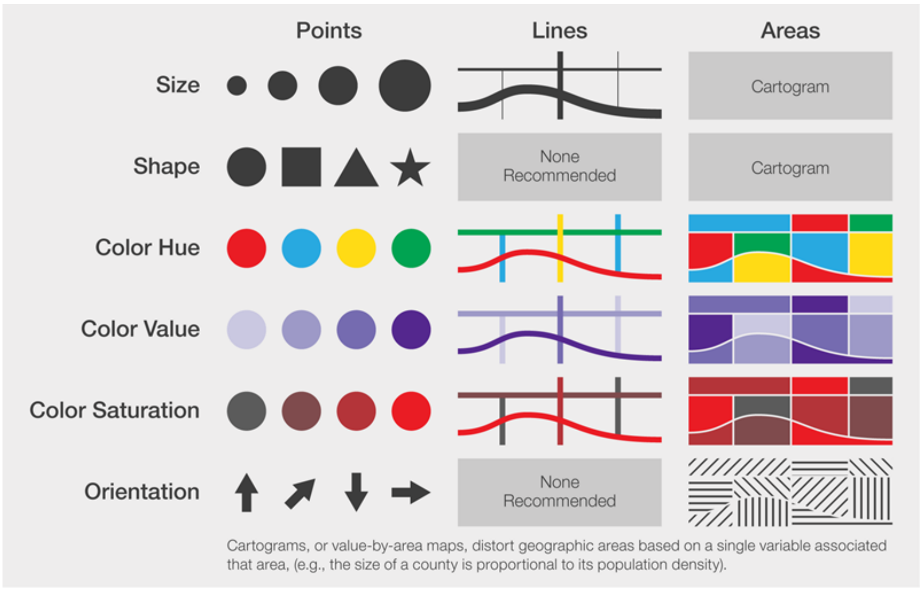
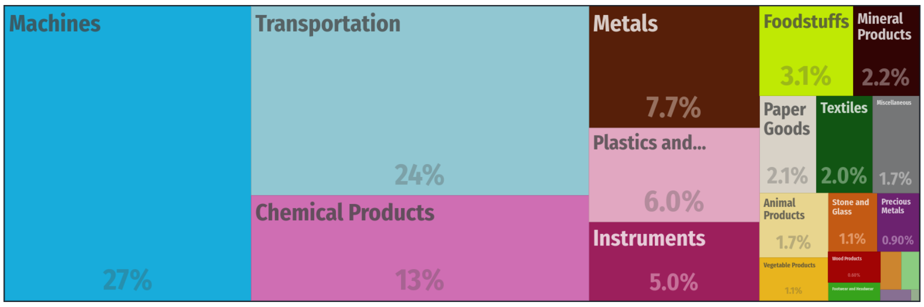

Points
Lines
Areas
Size
Shape
Color Hue
Color Value
Color Saturation
Orientation
Cartogram
None Recommended
Cartogram
None Recommended
Cartograms, or value-by-area maps, distort geographic areas based on a single variable associated that area, (e.g., the size of a county is proportional to its population density).

Fig. 2.1: Bertin's [5] original visual variables.

more than one node and soon exceeds the entire screen size. Johnson and Shneiderman [25] proposes the tree map visualization technique, in which each node is a rectangle whose area is proportional to a specified dimension. Rectangles contain sub-branches of the node as tiles, thereby expressing hierarchy information.



Ⓘ ⓢ Observatory of Economic Complexity [20]

Fig. 2.2: First hierarchy level of German exports

Why is the layout different in the second image?

We can see an example of an interactive treemap in Figures 2.2 and 2.3. German exports are divided in generic groups like "Machines" and "Chemical Products"

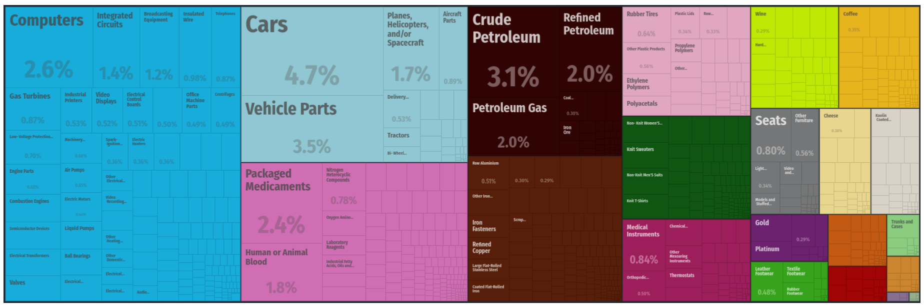


Fig. 2.3: Second hierarchy level of German exports

and include more specific groups like "Cars" and "Packaged Medicaments". The level of hiearchy can be selected with a dropdown menu, so only leaf nodes are displayed at a time. Treemaps are space-filling visualizations, i.e. they make 100% use of the available screen size.

Note that as the order and placement of the nodes depends on the value of their specified dimension, geographical units of data may may be placed separately on the treemap.

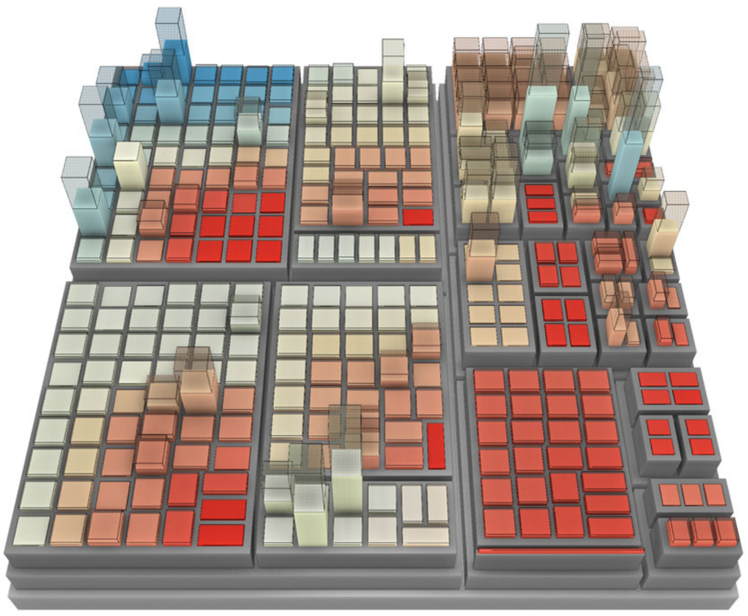
3D tree maps is a concept introduced by Bladh, Carr, and Scholl [8] in 2004. The authors transfer the concept of tree maps from two dimensional into three dimensional space. They introduce StepTree [8], which is a three dimensional tree map to display a directory layout. It "differs from Treemap in that it employs three dimensions by stacking each subdirectory on top of its parent directory." 3D tree maps are superior to 2D tree maps for tasks with a pronounced topological challenge. User perform significantly better in interpreting the hierarchical structure. However, 3D visualizations also introduce some disadvantages as superimposition of objects and a complex view point navigation.

2.5D tree map is a term coined by Limberger et al. [31]. A 2.5D tree map emphasizes physical constraints of a 3D tree map, i.e. a 2.5D tree map has all items attached to the ground. We can see an example of a 2.5D tree map in figure 2.4

2.1.4 Geographical Data Visualization

introduction

Flow maps place stroked lines on top of a geographic map. They often display the flow of goods or the migration of people. As we can see in Figure 2.5, an



© Hasso-Plattner-Institut[14]

Fig. 2.4: Example of a 2.5D tree map

early example of a flow map is Charles Minard's depiction of Napoleon's ill-fated march on Moscow in 1861 [12].

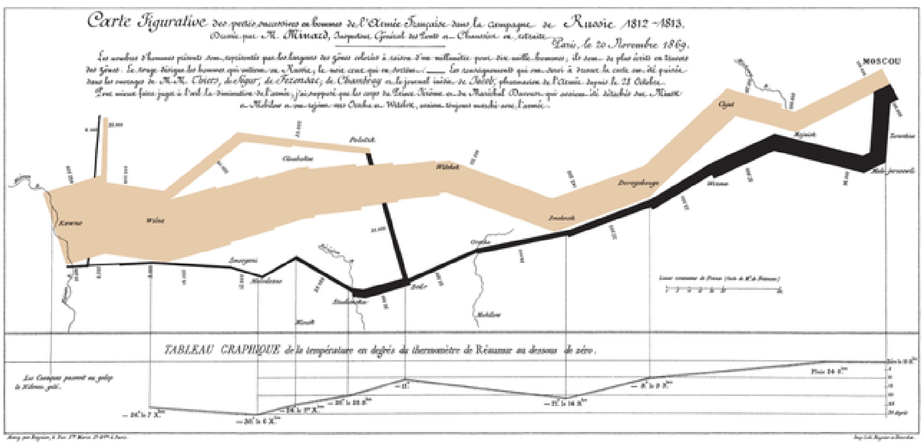
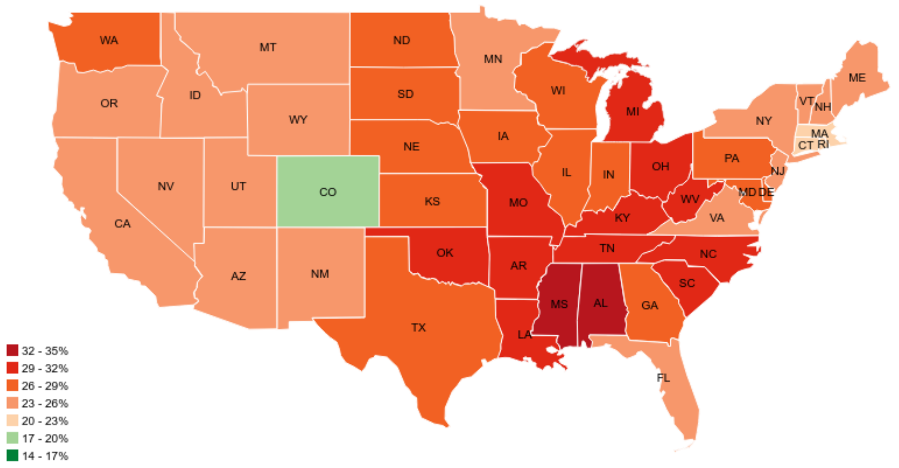

Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812–1813.
Dressée par M. Minard, Inspecteur Général des Ponts et Chaussées en retraite.
Paris, le 20 Novembre 1869.
MOSCOU
Kowno
Wilna
Smolensk
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.

Fig. 2.5: Charles Minard's map of Napoleon's disastrous Russian campaign of 1812. Minard managed to represent six values in two graphical dimensions: The number of Napoleon's troops, the travelled direction and distance, latitude and longitude relative to specific dates and the temperature.

Choropleth maps is a thematic map in which areas are shaded or patterned in proportion to the statistical variable being displayed on the map. A popular use case is the display of population density or per-capita income. We can see an example of a choropleth map in Figure 2.6, showing the percentage of obese population in the US. Choropleth maps are widely popular and likely to understand them. They are very helpful when data is attached to enumeration unites like counties, provinces and countries.



National Center for Chronic Disease Prevention and Health Promotion [**NCCDPHP2017**]

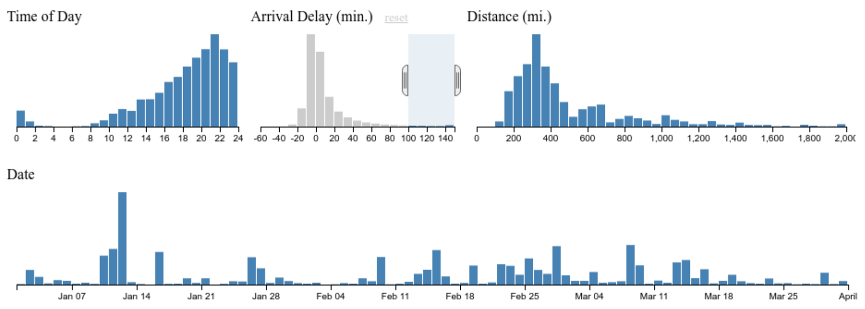
Fig. 2.6: Choropleth Map of Obesity in the United States in 2008. Present of population classified as "obese" (Body Mass Index in excess of 30), by state.

Graduated Symbol Maps is a good alternative to a choropleth maps, as it alleviates one of its downsides: Larger areas may appear more emphasized than smaller ones, thus confounding geographic area with data values. Instead, symbols are placed over the regions of a map. As Heer, Bostock, and Ogievetsky [21] point out, these symbols allow for more visal variables to be visualized, i.e. symbol size, shape and colour.

Cartograms are used to encode a data attribute in the size of an area. For this reason geographic regions in cartograms appear distorted. A common example is to redraw the size of a country based on the the population or gross domestic product of a country. Area cartograms may be contiguous or noncontiguous, depending on the applied layout algorithm.

2.1.5 Coordinated Multiple Views

According to Roberts [37] coordinated multiple views is just “a specific exploratory visualization technique that enables users to explore their data”. coordinated multiple views are characterized by the fact, that they show multiple views side-by-side. We can see an example in Figure 2.7. It displays an on-time performance of airlines, visualized with the “Crossfilter” javascript library. The user can set the borders of an interval with the mouse in each of the views. The visualization takes the most recent 80 flights from the database that match all given filters. All visualizations are then updated in real time. As we can see in the example in Figure 2.7 there seems to be a correlation of a long delay with a later time of the day.



Crossfilter[9]

Fig. 2.7: Airline on-time performance: Correlation of time of day with arrival delay. Most recent flight with a delay of more than 100 minutes selected.

Brushing and Linking Most multiple coordinated views also provide some kind of brushing technique. “The technique of brushing is the principle approach, where elements are selected (and highlighted) in one display, concurrently the same information in any other linked display is also highlighted.”[37]

2.2 Interaction Theory

According to Ho [22] interactions are a crucial part of data visualizations, yet most research in the area of data visualization still focuses on visual representations. Roughly speaking, research on interaction falls into these groups: How to categorize interaction techniques? How to find new interaction techniques and apply those to visualizations?

Give a rough overview of this section

2.2.1 Interaction Categories

Shneiderman [39] classifies interactions into these groups: (1) Gain an *overview* of the entire collection, (2) *zoom* in on items of interest, (3) select an item or group and get *details* when needed, (4) view *relationships* among items, (5) keep a *history* of actions to support undo, (6) allow *extraction* of sub-collections and of the query parameters.

In 1997 Shneiderman [39] classified interactions into these groups: (1) Gain an *overview* of the entire collection, (2) *zoom* in on items of interest, (3) select an item or group and get *details* when needed, (4) view *relationships* among items, (5) keep a *history* of actions to support undo, (6) allow *extraction* of sub-collections and of the query parameters.

Two years later, Dix and Ellis [13] identified these categories: (1) *Highlight and focus* particular subsets of the data, (2) instead of displaying everything simultaneously *access extra information* by drilling down the data, (3) zoom in and out to give an *overview and context*, (4) *change parameters* of the *same representation*, e.g. another baseline of a stacked bar char, (5) *change representation* of the *same data* by switching the chart type, (6) *link representations* to determine the relationship between items.

In 2002, Keim [27] comes up with the following classification: (1) Dynamic *projection* to show all combination of data attributes mapped to the axis of a diagram, (2) focus on a smaller subsets by *filtering* out parts of the data, (3) *zoom* into a subset of the data and get a higher level of detail, (4) preserve an overview of the data during drill-down operations is called *distortion* (5) and finally *link and brush* visualizations, to highlight the same data points in multiple visualizations.

The most recent classification was done in 2007 by Yi, Kang, and Stasko [44] listing seven categories: (1) *Select* to mark something as interesting, (2) *explore* to show something else, (3) *reconfigure* to show a different arrangement, (4) *encode* to show a different representation, (5) *abstract/elaborate* show more or less detail, (6) *filter* show something conditionally, (7) *connect* show related items.

The classifications are all redundant! Explain why and choose one classification for later use

Give one example for each category of the chosen classification

2.2.2 Interaction Models

no section without text

Space-Time Cube Operations is a concept introduced in 2014 by Bach et al. [4] to map temporal data into two dimensional visualizations. Space-time-cubes are used to model two attributes of continuous data with temporal data along a third axis, therefore the name *cube*. While the transformations are rather static it is also possible to introduce activity into the transformations. The authors describe user-independent *animations* and user-controlled *interactions*. E.g. a transformation may display a given slice of the cube. An animation would display one slice at a time and display the next slice every second. Whereas the interaction would show the slice determined by a user-controlled slider. Various transformations and their best use in practice are evaluated in this work. The work focuses on temporal data and otherwise continuous data. Interactions are not seen as an abstract entity, that need to be agnostic of the underlying data structure and visualization. The authors admit “our framework does not provide much guidance for interaction design: the design space for interactive operations has only been partially explored.”[4, Other limitations, p. 15]

ITlib[17] is an architecture and a framework of interaction techniques for virtual reality applications, designed to be extensible and flexible. New interaction techniques can easily be added and application specific code is seamlessly integrated. On a low level an interaction technique “is modeled as a set of filters connected in a small data flow”[17, Basic concept, p. 2]. These filters are the smallest process unit in the data flow. Composed of input and output ports, they communicate with other filters, to receive data input from predecessors and send data output to successors. The framework specifies and stores the interaction techniques along with its filters, the execution model and the scene in XML documents. The authors chose XML because it can be parsed easily and they generate code in order to target various virtual reality toolkits and environments.

Even though the system describes interactions in an abstract way, the domain of the framework is clearly the interaction of a human body within a 3D virtual reality. Certain assumptions are made, including the data model, which is the 3D scene, and human computer interaction devices, like the user's hand or the user's head. The goal is not to better understand the data, as the data model in this case is the 3D scene, and not statistical data. Most important, the framework describes interaction techniques for a single viewpoint but not for coordinated multiple views.

Focus+Context Visualization by Bjork, Holmquist, and Redstrom [7] is one of the few formalizations of information visualizations. The authors describe this formalization based on first-level and second-level visualization:

Visualizations referred to as IV, are triples of a set $[D]$ of underlying data, a visual representation V and I which is the possible interaction or manipulation.

$$IV([D], V, I) \tag{2.1}$$

If I affects $[D]$ we can manipulate the underlying data set. Examples would be changes in a spreadsheet editor, or a change of the start and end date of an appointment in a calendar. When V is affected by I the user can manipulate IV in order to change the way $[D]$ is represented, e.g. choosing a different level of detail as shown in Figures 2.2 and 2.3.

Second-level Visualizations are information visualizations consecutively applied. The underlying data set $[D]$ of the previous formula is replaced with some information visualization IV, which is compatible with IV'.

$$IV'(IV, V', I') \tag{2.2}$$

Focus+context visualizations are second-level visualizations. An example given by the authors is the “rubbersheet” visualization, that visually distorts a first-level visualization similar to a magnifier.

3

Analysis

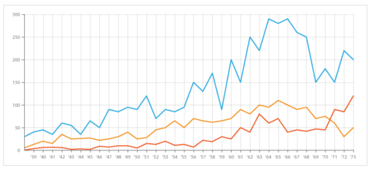
One purpose of this section is to abstract and deduce the essential characteristics of any interaction. These characteristics are relevant for a formalized language of coordinated multiple views. To accomplish this goal, we take the approach of deducing the characteristics by example. We analyze each data visualization example in terms of the expected data structure and dependent and independent visual variables [5]. We suggest interactions for each visualization as well as interactions that run on multiple views. Those interactions will be classified according to Yi, Kang, and Stasko [44] and the relevant subject of the interaction is specified.

The second outcome of this section is a set of requirements which can be used to evaluate a framework of coordinated multiple views.

3.1 Single Visualization Interactions

The data visualization catalogue by Severino Ribecca list many of the most used data visualizations[36]. Let's pick some of those and derive a number of possible interactions.

Line diagrams



(a) Line diagram

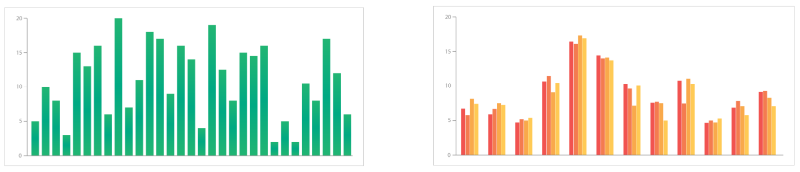
Fig. 3.1: Line charts are used to display trends

Line diagrams are multiple sets of data, displayed along the x-axis. They can display a quantitative value over a continuous interval or time span and often they are used to show trends. It is possible to highlight an entire series of data or just a feature within that series. Line diagrams expect data in a tabular format, a few columns with many data points each. The dependant visual variables here are (1) Position (2) Orientation and (3) Texture.

Table 3.1: Interactions for line charts

Select	Highlight a data point (id of data point)
Select	Highlight a data series (id of data series)
Encode	Change colours of data series (data series → colour)
Filter	Restrict interval on x-axis (filter function of data attribute)
Filter	Hide a data series (id of data series)

Bar charts and multiset bar charts



(a) Bar chart (b) Multiset bar chart

Fig. 3.2: A multiset bar charts is a variation of a bar chart

Bar charts and multiset bar charts show one or many attributes per feature along an axis. They have in common that they encode the data attribute into the height of the eponymous bars. Colors might be used to better distinguish between the different data attributes. Features can be arbitrarily ordered along the axis, although multiset bar charts group features along a series of categories.

Therefore, bar charts need to be initialized with the set of features and their values as well as the groups of features for the multiset bar chart. The supplied colours would colour each feature in a group in turn. The set of possible interactions include the highlighting of features, the reordering of features and a change in encoding of colour values.

Data structure	Tabular data, many data sets as series
Dependent visual attributes	Size, orientation.
Independent visual attributes	Position, colour, shape, texture.

Table 3.2: Interactions for bar charts

Select	Highlight a bar (id of data point)
Encode	Change colours of data series (colours $\rightarrow$ data series)
Reconfigure	Sort by attribute (data attribute)
Reconfigure	Drag bars to reorder data series (ordered list of ids of data points)
Filter	Hide a data series (id of data series)

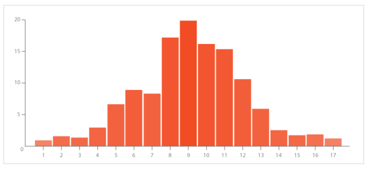


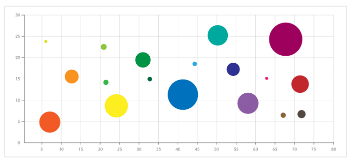
Fig. 3.3: A histogram is a bar chart over a continuous interval

Histograms visualise the distribution of data over a continuous interval or certain time period. A special type is the population pyramid, which is a pair of back-to-back histograms, one for each sex. The difference of histograms to bar charts lies in the type of data itself, not the representation. Therefore these charts need to be initialized with the same data and the same interactions can be applied.

Data structure	Tabular data, many data sets as series
Dependent visual attributes	Size, orientation, position.
Independent visual attributes	Color, shape, texture.

Bubble charts and scatter plots

Bubble charts are popular choices to display a distribution of features. The chart is initialized with coordinates for each feature, a colour and a size in case of a bubble charts. Possible interactions include the highlighting of features, a different colour encoding, a reconfiguration to map another attribute to size. Bubble charts may show only a window of the available data and allow to zoom in, zoom out or move the window along the axes.


(a) Bubble chart

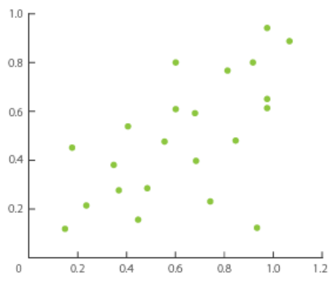

(b) Scatter plot

Fig. 3.4: Bubble charts and scatter plots are similar regarding interactions

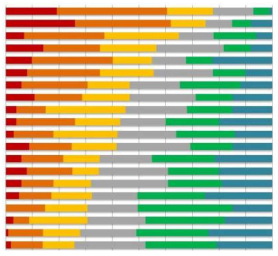
Data structure	Tabular data, single data set with x- and y-coordinates
Dependent visual attributes	Size, position.
Independent visual attributes	Color, shape, texture, orientation.

Table 3.3: Interactions for bubble charts

Select	Highlight a bubble (id of data point)
Explore	Zoom in, zoom out (width and height of window)
Explore	Move viewport position (x- and y-coordinates of viewport)
Encode	Change mapping of colour to category (data series → colour)
Encode	Change colour function (function value → colour)
Encode	Change data attribute to colour (data attribute)
Encode	Change data attribute to size
Reconfigure	Sort by attribute (data attribute)
Reconfigure	Drag bars to reorder data series (ordered list of ids of data points)
Filter	Hide a data series (id of data series)

Stacked bar charts

Unlike a multi-set bar graph which displays their bars side-by-side, stacked bar graphs segment their bars of multiple datasets on top of each other. A baseline, as shown in figure 3.5 might be modeled as two back-to-back multi-set bar graphs. A reordering would e.g. move one data set from the left side to the right side. Possible interactions include the highlighting of a feature, a change of color mapping, reordering of the baseline.


(a) Stacked bar chart

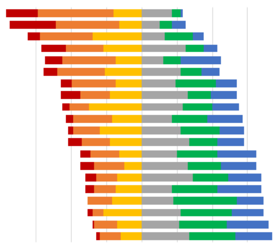

(b) Stacked bar chart with baseline

Fig. 3.5: Stacked bar charts can be ordered along a baseline or stretch to 100% width to show the percentage-of-the-whole of each group

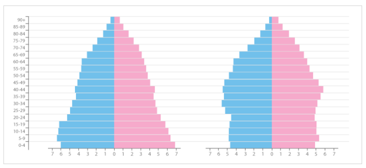


Fig. 3.6: A population pyramid can be modeled as a stacked bar chart

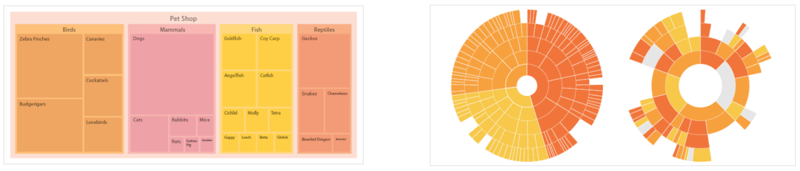
Table 3.4: Interactions for stacked bar charts

Select	Highlight a bar (id of data point)
Encode	Change mapping of category to colour (data series $\rightarrow$ colour)
Reconfigure	Sort by attribute (data attribute)
Reconfigure	Reorder Y axis (ordered list of ids of data points)
Reconfigure	Sort stacking order by attribute (data attribute)
Reconfigure	Specify the stacking order data series (ordered list of ids of data series)
Reconfigure	Specify a negative data series (list of ids of data series)
Filter	Hide a data series (id of data series)

Data structure	Tabular data, multiple date sets as series
Dependent visual attributes	Size, shape, orientation.
Independent visual attributes	Color, position, texture.

Hierarchical visualizations

Treemaps are great to show hierarchical data without ever exceeding the available screen. Each feature is a assigned a rectangle according to a layouting algorithm. Unlike a tree map a hierarchical ring diagram or sunburst diagram shows each level of the tree as a series of rings.



(a) Tree map (b) Sunburst diagram

Fig. 3.7: Tree maps and sunburst diagrams are ideal to show hierarchies

Therefore, both tree map and ring diagram need the feature set as a tree, with each node having a data attribute for layouting. Every node may be assigned a color. As we are describing hierarchies, the maximal depth of tree may be increased or decreased. Again, interactions could include a highlighting of features and a change of color encoding. Both visualizations may show only a subtree. E.g. a click on a box in the treemap opens another treemap focused on the subtree. Similarly a click on a slice of the ring would surround the most external ring with the children of the feature.

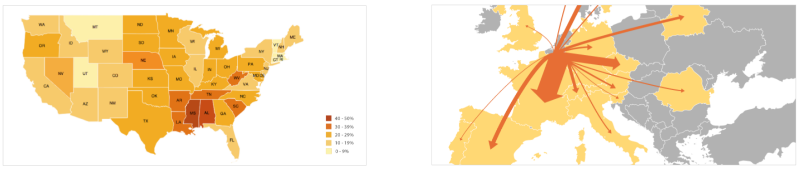
Data structure	Tree, each feature has a value for layouting.
Dependent visual attributes	Position, Size, shape, orientation.
Independent visual attributes	Color, texture.

Table 3.5: Interactions for hierarchical visualizations

Select	Highlight a feature (id of data point)
Explore	Use another node as root of the visible tree (id of data point)
Encode	Change mapping of category to colour (data series → colour)
Reconfigure	Change data attribute used for layouting (data attribute)
Reconfigure	Sort by attribute (data attribute)
Reconfigure	Specify order (ordered list of ids of data points)
Abstract/Elaborate	Specify maximum depth of visible tree (number of levels)

Geographical Data

Choropleth maps and flow maps are specialized diagrams focused on geographical data. Size, position and shape of a feature is defined by the geometry data of a feature. In choropleth maps the color of each feature is based on a data attribute. Flow maps may display connections between features, a data value defining the size of each arrow.



(a) Choropleth map (b) Flow map

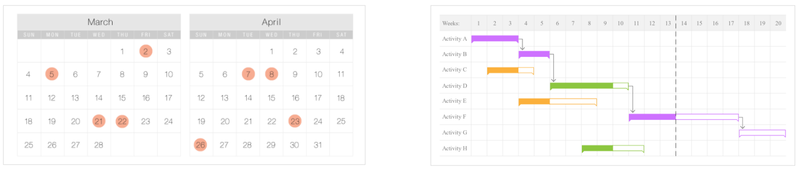
Fig. 3.8: Choropleth maps focus on a density while flow maps show a migration of data

Data structure	Graph data with edges, each feature has geometry data.
Dependent visual attributes	Position, Size, shape, orientation.
Independent visual attributes	Color, texture.

Table 3.6: Interactions for geographical visualizations

Select	Highlight a feature (id of data point)
Explore	Move viewport (latitude and longitude of viewport)
Explore	Zoom in, zoom out (zoom factor)
Encode	Change shape of marker (data id → shape)
Encode	Change mapping of category to colour (data series → colour)
Encode	Change colour function (value → colour)
Encode	Change data attribute used for colour (data attribute)
Connect	Show relations of a feature (id of data point)
Abstract/Elaborate	Change granularity of displayed regions (number of levels)

Activity diagrams



(a) Calendar (b) Gantt chart

Fig. 3.9: Similar to a calendar, a gantt chart shows activities and the progress along a time line

In activity diagrams, each feature is represented as a rectangle, with the duration of the activity mapped to size and position. Calendars and gantt charts could not only read the data from the data source, but also add new features to the data set or update metadata of a feature, e.g. the progress of the activity.

Data structure	Temporal data, each feature has a time interval.
Dependent visual attributes	Position, Size, orientation.
Independent visual attributes	Color, shape, texture.

Table 3.7: Interactions for temporal visualizations

Select	Highlight a feature (id of data point)
Explore	Show a different period of dates (start and end datetime)
Explore	Show a different time interval (start and end hour)
Encode	Change color of categories or activities (data series → colour)
Encode	Change data attribute used for colour (data attribute)
Filter	Remove a calendar or a category (id of data series)

3.2 Multiple View Interactions

Detail view

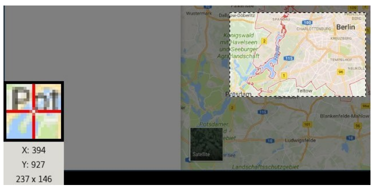


Fig. 3.10: The screenshot-tool “shutter” shows a magnified detail view of the area around the mouse cursor in the lower right corner of the screen

3.3 Requirements

In this subsection, we list a set or requirements imposed on a coordinated multiple view framework. These requirements can be used for further evaluation.

Serialization is the process of translating objects that can be stored or transmitted and reconstructed later. In order to coordinate interactions among views, information needs to be passed from one view to another. A framework for coordinated multiple views should therefore find a serialization format for interactions which has (1) small payloads and (2) fast serialization and deserialization.

Reversibility in the context of a coordinated multiple view framework means if it is possible to undo the effect of an interaction. Ideally, every interaction function should have a well-defined inverse. For every interaction that is not reversible, the computational cost to replay the interactions from the original state up to the point of the interaction should be minimal.

Data extensibility indicates the ease of reloading and updating data on the fly. This is especially important if an interaction requests additional data from an external service. We consider good extensibility when (1) additional data attributes can be added without lookup of corresponding items and (2) no deduplication steps are necessary when new items are added.

Development costs qualify how much time and effort is needed in order to develop new components for the coordinated multiple view framework. We track these costs in working days and the number of changed lines of code.

Maintainability means in our case, how much other views are impacted by a change of an interaction in one view and how error-prone the framework is. In general it is hard to measure maintainability. For the coordinated multiple view framework we want to measure the (1) lines of code and the (2) cycliomatic complexity. We will try to find a means to measure (3) cohesion and (4) coupling in the framework. We assume that the amount of shared data between views is an indicator for high coupling.

4

Concept

Based on the findings in Section 3 we specify a formal language and a conceptual framework for coordinated multiple views in this section. Our approach involves the definition of the terminology and the basic components of the conceptual framework. The first component is the shared data model on which all visualizations must agree on. We formalize each interaction from Section 3 as a mathematical function and group these functions together if they have the same input and output. As a last step, we specify the communication pattern of the formalized language.

4.1 Terminology

We introduce a couple of new terms in order to give a formal definition of the conceptual framework

Trigger describes the event that starts an interaction. As we have seen in Section 3 every interaction is started by a user action. The user e.g. clicks on a shape in the view, hovers over an area, selects an item from a dropdown menu, turns around a mobile device, speaks into the microphone or makes a particular gesture. If the event of action is handled by a view and causes an interaction, we call this handling a *trigger*. Every view is responsible of its *triggers*. These are implementation details and are not shared by other views.

Effect is the change of the visual representation subsequent to the interaction. In order to be perceivable by the user, the interaction must have some visual effect, i.e. a change in the visual representation of the data. Examples are: A change of colour of a selected bubble, a movement of the viewpoint, a rearrangement of attributes in a parallel plot or a higher level of detail in a 2.5D tree map. Every view is responsible of its visual *effects*. Those are implementation details as well and they are not shared by other views. Obviously visual effects are not shared, as it depends on the visualization if visual variables are constrained or available to express a visual effect.

In singular visualizations, interactions consist of at least of a trigger and an effect:

$$I(T, E) \tag{4.1}$$

The meaning of the interaction in a singular view can be implicit. E.g. hovering over a geographic area changes the background colour of the polygon and the user can identify the interaction as a *highlighting*. Note that this implicit meaning must be explicit in coordinated multiple views so that other views are able to process it.

Subject refers to the target of the interaction. We must define what data or meta-data is affected by the interaction. E.g. when a user moves the mouse cursor on a line in the line diagram, that could highlight the *data point* under the cursor as well as the entire *data series*. Therefore we call the object affected of an interaction the interaction *subject*. A subject can be a data point, a list of data points, a position of the viewpoint, a certain order of attributes or a mapping of attributes to visual variables.

Verb refers to the application specific context and purpose of the interaction. A developer may want to have many *select* interactions. E.g. the user desires to select a detail view of an item under the mouse cursor from a previously selected set of already highlighted items. Therefore we allow for a user-defined *verb*. The verb describes the interaction in the context of a task or an intention that is intrinsic to the application the interaction happens in.

Interaction function is the declaration and definition how the subject of the interaction should be changed. The aforementioned explicit meaning of the interaction is the smallest unit of information of the interaction. This meaning is shared among multiple views, it is agnostic of how the interaction is implemented. Therefore the meaning is shared between multiple views. We call this meaning the *interaction type*. Some examples of interaction types include: Selection, Deletion, Point-of-Interest, Filtering, Reordering, Re-encoding. Interaction types can be classified with the interaction categories of Yi, Kang, and Stasko [44].

Note that different information visualizations implement an interaction types in a different way or maybe not at all:

E.g. a bubble chart might encode a certain data attribute to the colour of a bubble. Therefore the colour would not be changed for an interaction of type *Selection* but the texture of the bubble.

A user may perform an interaction of type *Reordering* on the series of a stacked bar chart. This can have no effect on a coordinated scatter plot, as the position of each data point depends on its dimension.

Sentence is the minimal set of information to coordinate an interaction across multiple views. We define a sentence S as:

$$S(V, F(IS)) \tag{4.2}$$

With V as a verb, an interaction function F and the interaction subject IS. See Section 4.5 for a list of interaction functions. A sentence contains everything necessary to ensure the interaction can be reflected in another view.

Application context is name for the state which is exclusive to the specific application. Part of the application context is the access to data sources and the procedure to initialize data visualization with it. Also, the binary files of textures, icons and vectors that describe the visual representation of a shape and colour themes are part of the application context.

4.2 Formalization of an interaction

We formalize an interaction I as follows:

$$I(T, S(V, F(IS)), E) \tag{4.3}$$

For a trigger T, a sentence S and the effect E. The sentence S is composed of a verb V, an interaction function definition F and an interaction subject IS. The part of the interaction which is shared among multiple views is the sentence S. The function definition F is defined by the triggering view at runtime. E.g. when a parallel plot rearranges the list of attributes, the exact order will be determined by the view itself, during the handling of the drag-and-drop action.

4.3 Conceptual Framework

no section without text

4.4 Shared data model

To account for the various data structures, we use an abstract data model that is powerful enough to include tabular, hierarchical and relational data. You can see a class diagram in figure 4.1.

The *entity* class is used to model the smallest distinguishable unit. All entities can be identified and retrieved via the *id*. An entity is defined to be any object that can have data attributes attached as *dimensions*.

While entities describe what an object *is*, a *dimension* describes what it *has*.

An entity can have arbitrary many attributes and each value can be accessed by the name of the attribute. So if you want to get the *latitude* value of an entity, you can retrieve the value with a call to the dimension *latitude*.

Entities can also be *series* of other entities. A series contains an ordered list of contained entities. As series can also contain other series, so we can model a hierarchy relation.

Every entity has a *parent* which is the series it is contained in. The root entity of the hierarchy has a parent which is `nil`. Every series has a special attribute `height` that describes the number of nested series or the height of the subtree.

If we just want to display tabular data, we just have one or two levels of hierarchy. E.g. one level of hierarchy for a histogram and two levels of hierarchy for a stacked bar chart.

Other relations than hierarchical relations can be modeled as a *relation* entity. It represents a directed edge in a graph and must have incoming and outgoing entity. Since every *relation* is an *entity* as well, we can add *attributes* to the relation. These attributes may describe e.g. the weight of an edge in a flow map.

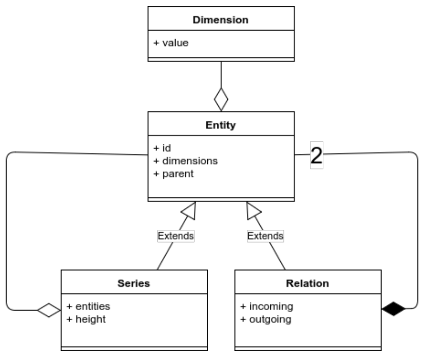


Fig. 4.1: A data structure for tabular, hierarchical and relational data

4.5 Predefined Interaction Functions

It turns out that we can describe the types of an interaction as a mathematical function. These functions operates on the ids of *entities*, *series*, and *relations* or their respective *attributes*. The name of the function is the *type* of the interaction and the function domain being the *subject* of the interaction. Thus, we can describe interaction through a change of its semantic but we ignore implementation details of specific visualizations.

These function are derived from specific interactions of the examples in Section 3. Domain and range of these functions refer to the objects defined in the data model in Section 4.4. To declare the functions explicitly, we define a couple of sets:

$$\mathbb{E} : \mathbb{E} \subseteq \mathbb{N} \tag{4.4}$$

The set of all entities in our subject space. Each entity can be represented by its `id`, so for simplicity $\mathbb{E}$ is a subset all natural numbers in $\mathbb{N}$.

$$\mathbb{D} : \mathbb{D} \subseteq \Sigma^* \tag{4.5}$$

The set of all dimensions in our shared data model. Dimensions of a data set are the attributes of our data model and both terms are used synonymously in the following.

$$\mathit{Space}(\mathbb{D}) \tag{4.6}$$

Where *Space* is the range of values of a dimension $d \in \mathbb{D}$.

For convenience, we define this set with a function *Space* which maps the name of a dimension d to its set of possible data values. So for an attribute `name` that would be the set of all strings. For continuous values, that would be the set of all real numbers $\mathbb{R}$. But $\mathit{Space}(D)$ also includes the set of all possible discrete values of dimension.

$$\mathbb{V} = \{x, y, y2, y3 \ldots yn, z, \mathit{height}, \mathit{colour}, \mathit{size}, \mathit{shape}, \mathit{orientation} \ldots\} \tag{4.7}$$

A discrete set to capture visual variables position, size, orientation, texture, colour and shape, as described in Section 2.1.2. It is required to allow for *encoding* interactions, that change the mapping of a dimension to a visual variable.

Note about notation: The power set of all entities in E is written as $\mathcal{P}(\mathbb{E})$ and is the set of all subsets of E. The list of all sequences of all entities in E is written as $\mathbb{E}^*$ and includes all enumerations of $\mathcal{P}(E)$. The empty set is written as $\varnothing$. If a function has the empty set as its domain, it expects no arguments. Such a function is also called a constant.

The following is a list of functions describing the types based on the aforementioned sets:

$$Select : \varnothing \to \mathcal{P}(\mathbb{E}) \tag{4.8}$$

The constant *Select* takes no arguments and returns a subset of selected entities. A *Select* can be used to highlight entities or to show details of entities. It can be used to mark entities for deletion or to temporarily hide entities. In addition it possible to focus the visualization on these entities.

An example of the latter: A geographical map could move the viewpoint position and the zoom level such that all focused entities in *Select*() are visible. Hierarchical visualizations, e.g. tree maps or sunburst maps, may choose a single focused entity to be the root node of the currently displayed subtree.

$$Filter : \mathbb{E} \to \{\bot, \top\} \tag{4.9}$$

The *Filter* function describes which entities are part of the visualization. It can be implemented explicitly or implicitly. An explicit implementation would returns `true` or `false` based on the fact if an entity is part of an already known set. Implicit implementations would be based on the values of the dimensions of the entity. A threshold function is a good example of an implicit implementation.

$$Window : \mathbb{D} \to \mathcal{P}(Space(\mathbb{D})) \tag{4.10}$$

This function defines the currently visible section of the vector space of the dimensions in $\mathbb{D}$. For each of the dimensions in $\mathbb{D}$, the function returns the currently visible subset.

The subset can be defined implicitly or explicitly: For charts with continuous values along the x- and y-axes, the function returns the representatives `from` and `to`. These representatives implicitly define the interval in $Space(\mathbb{D})$. If a dimension has discrete values, this function can also return an explicit set of values.

Scatter plots, bar charts, line diagrams, bubble charts all have two coordinate axes. Therefore `x` and `y` will each be mapped to a pair of values $(from, to) \in \mathbb{R}$. Geographical visualizations have the camera pointing to the center of the earth. We have three degrees of freedom, so we would map `latitude`, `longitude` and `zoom` to three values in $(lat, long, z) \in \mathbb{R}$. In a calendar we map `fromDay`, `toDay`, `fromHour` and `toHour` to define the currently visible time section. A special attribute of our shared data model is the height of the subtree of an entity, i.e. how many nested series we have below that entity. Therefore we can also map `height` to a single value to define the maximum depth of the currently visible subtree in a tree map.

$$Order : \mathbb{E} \times \mathbb{E} \to \mathbb{R} \tag{4.11}$$

The *Order* function is used to order two arbitrary entities $e1, e2 \in \mathbb{E}$. If $e1 < e2$ then the return value r will be $r < 0$. For $e1 = e2$ the statement $r = 0$ holds and for $e1 > e2$ then $r > 0$.

Similar to *Filter*, the *Order* function can either be implemented explicity or implicitly. An explicit implementation returns a value r based on the relative position of $e1$ to $e1$ in a given sequence. An implicit implementation would return a value r based on some computation of dimensions of $e1$ and $e2$. E.g. ordering entities based on the alphabetical order of their name would be an example of the latter.

$$Encode : \mathbb{D} \to \mathbb{V} \tag{4.12}$$

The *Encode* function can be used to change any mapping of dimension to any visual variable. E.g. bar charts, line diagrams, histograms and bubble charts can change the attribute mapped to the `y` and `x` axes. Bubble charts can encode a different data attribute in the `size` of the bubbles. Choropleth maps, treemaps and bubble charts can map a different attribute to the `colour`. A specialized version of this function may return the attribute that is used for the `layout` of the tiling algorithm in tree maps. Note that parallel plots have arbitrary many `y` axes. To define the order of dimensions displayed in a parallel plot, each dimension will be mapped to a named `y` axis, e.g. $y1, y2...y3$ and so on.

Let's have some examples how these functions can be applied on coordinated interactions:

A user clicks on a bar in a bar chart and this feature then changes its background colour. To coordinate the highlighting, the bar chart view will formulate a new sentence composed of a *Select* function and the verb *highlight*. The function returns the set with the highlighted entity.

Let's say, a geographical map should move the position of the viewpoint on a entity. The triggering view will use a function *Select* this time with a verb *focus*. A treemap as a third view could pick up that interaction and show a subtree with the focused entity as root node.

A view may show some controls to filter the data set, e.g. two sliders on an attribute called `prize`. When the user releases the mouse, an interaction with the function *Filter* and the verb *Hide* will be triggered. The function will then check the `prize` of every entity and returns `true` if the prize is within the given lower and upper limit, `false` otherwise.

4.6 Communication pattern

In Section 4.5 we discussed how we can describe the types of the interaction. The interaction type determines the signature of the function and describes the interaction itself. However a type does not define how interactions are *coordinated* among views. E.g. what action in one view should lead to what kind of changes in what other views? What is the communication pattern or what is the protocol how sentences are exchanged in the conceptual framework?

Figure 4.2 gives an overview on the message flow in the communication pattern.

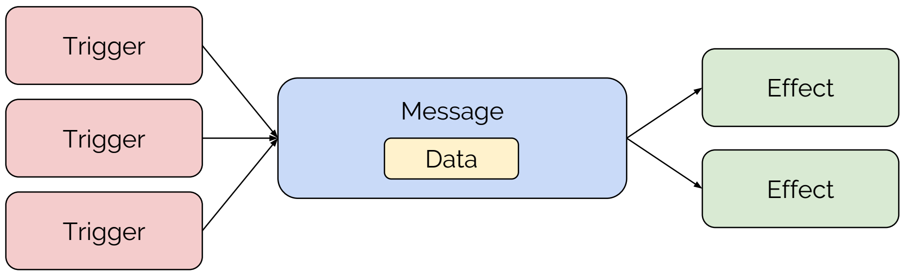


Fig. 4.2: The message flow of the communication pattern: Every view can decide for itself whether to trigger an interaction or respond to an interaction with a visual effect.

Sometimes a visualization may not be able to interpret an interaction. E.g. a bar chart can re-arrange the bars along the x-axis in case of a *Reconfigure* interaction.

But a scatter plot constrains x- and y-coordinates of an entity on a certain data attribute. Therefore, not only the *trigger* and *effect* is implementation specific, but also the handling of the interaction itself.

Every visualization decides on its own, how to react to a certain interaction. That leads us to distinguishable, named interactions. Every view can subscribe to certain interactions and receive messages in form of changed interaction types. In order trigger an interaction, the visualization simply publishes to the named interaction.

This pattern is known as the *Publish-subscribe pattern* and widely used in message queues. The term *interaction* in our case is equivalent to the term *channel* or *topic* commonly used in message queues.

5 Implementation

This section describes the reference implementation for the conceptual framework in Chapter 4. First, the general architecture is explained. Various components and interaction techniques of the geographical visualization are detailed. The integration of the geographical visualization in the existing VISUAL ANALYTICS PLATFORM is shown. The connection of the conceptual framework and the implemented framework is examined.

5.1 Architecture

The reference implementation of the conceptual framework is written in `TypeScript` and uses `PubSubJS` for communication. The visualizations are implemented with `React` and the geographical visualization uses `Leaflet` under the hood. Geometry data is exchanged with `GeoJSON` as data format.

TypeScript TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It provides optional static typing, classes and interfaces.

PubSubJS is a topic-based publish/subscribe library written in JavaScript [38]. Topics can be registered hierarchically, with subtopics delimited by dots. A subscription to topic `mcv.select.focus` will be notified only for `focus` interactions whereas a subscription of `mcv.select` will be notified for both `focus` and `highlight` interactions. Furthermore, topics are published asynchronously, so if the user interacts with a visualization, that does not block code execution.

GeoJSON GeoJSON is a format for encoding a variety of geographic data structures [1]. Based on JSON, it can simple geographic features like points, lines and areas and reserves a properties object for non-spatial attributes.

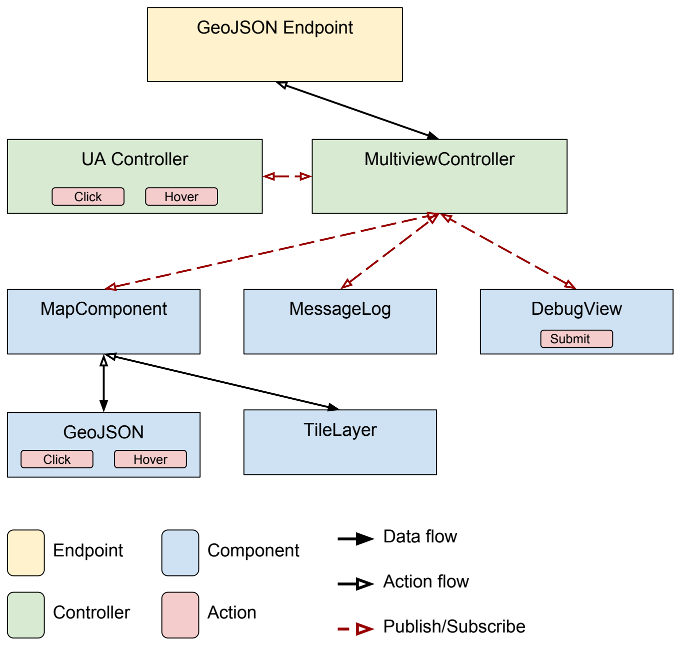


Fig. 5.1: Architecture of the reference implementation.

Leaflet Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps [3]. It has support for GeoJSON which makes it very easy to display tiled web maps with interactive overlays.

ReactJS ReactJS is an open-source JavaScript library to build user interfaces and allows to create reusable UI components. React renders HTML on the client, it changes the page without reloading the page. The framework corresponds with the View in the Model-View-Controller pattern. React components are structured hierarchically, with each component having dedicated responsibility.

The general architecture of the reference implementation is shown in Figure 5.1. Within the existing VISUAL ANALYTICS PLATFORM the `UAController` controls the rendering of the 2.5D tree map. The rendering of the geographical visualization is controlled by the `MapComponent`. Additionally, there are two other views, `DebugView` and `MessageLog`. These views record interactions at runtime and can trigger and simulate an interaction.

Every view has a reference to `MultiviewController` in order to subscribe to interactions. The `MultiviewController` itself does not have a visual representation.

The classes `MapComponent`, `DebugView` and `MessageLog` are implemented with React and have a **`render`** method. This way, these class can be initialized and rendered anywhere in the DOM tree. They are dependency free, except of the reference to a `MultiviewController`, which makes those classes easy to test.

The views that are implemented with React automatically re-render if their internal state changes. Event handlers just have to publish an interaction, if the view is subscribed to the same interaction, it will automatically re-render.

5.2 Software patterns

Figure 5.2 shows how coordinated multiple views can be automatically updated even if the environment lacks a native update-mechanism.

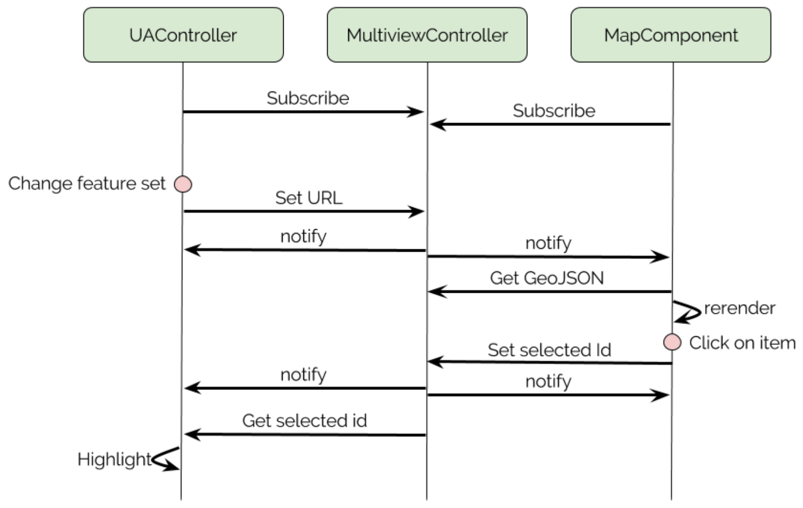


Fig. 5.2: The sequence diagram shows the notification of different components. The user first chooses a feature set and hovers over a polygon in the geographical map.

The Observer pattern allows multiple *observers* to react to changes of an observed state. In our case, the observed state is the `MultiviewController`. Any change to the `MultiviewController` will subsequently be broadcasted to all connected views.

Publisher subscriber In our particular case we apply a special form of the observer pattern, the so called “Publish-subscribe” pattern[15]. Publish-subsribe is a messaging pattern which is widely used in message queues. In this scenario, senders of messages simply categorize their messages which will be consumed by subscribers of the category. The scenario has very low coupling, publishers do not even need to know the existence of subscribers.

Component pattern State-of-the-art javascript component frameworks like ReactJS and EmberJS follow the component pattern for the architecture of a single page web application. The component pattern imposes a hierarchical structure on a website. Each component is responsible for a task and may contain other components. The components are joined at the root node of the page.

This pattern is very applicable to coordinated multiple views. The different views of coordinated multiple views share state, i.e. the feature, that is currently highlighted or the applied filter on the data. So the views are components and their closest common ancestor is the coordinated multiple view itself, controlling state and passing user interaction down to it’s children.

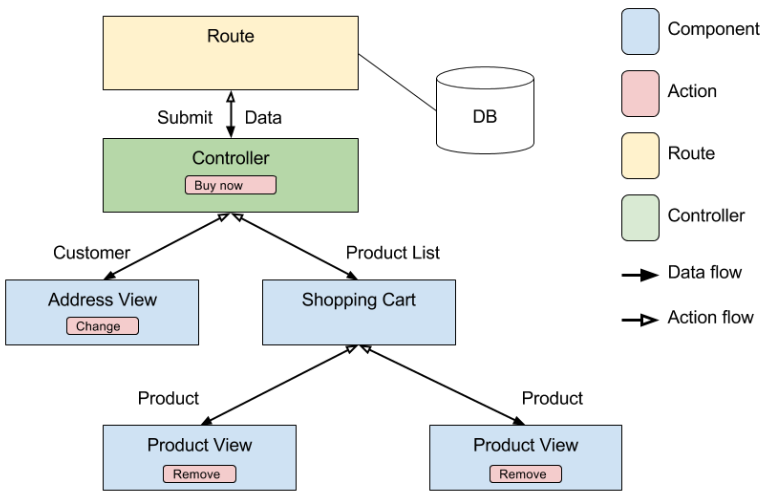


Fig. 5.3: Implementation of the component pattern in EmberJS. The example shows a page of a webshop. The customer is about to order the items in the shopping cart.

Actions up — Data down

Version 2.0 of Ember introduced a common phrase how to use this pattern effectively: “Data down, actions up”[41] In the domain of coordinated multiple views actions would mean user interactions, e.g. a click on a feature. The action will notify the controlling coordinated multiple view component. Actions may change data, and the changes will be passed to to all dependen views. These views are then rerendered.

Examples for the kind of data that might trigger a rerendering of a view:

- The selected feature or a list of selected features
- A list of thresholds for certain features as a filter

5.3 Geographical visualization with React and Leaflet

The geographical visualization is implemented as a `Map` component of the React Leaflet library. Listing 5.1 shows the `render` method of the component.

```
render() {
  return (
    <div className="multiview-map-component">
    <Map ref={(m) => this._map= m} bounds={this.bounds()}>
    <TileLayer
    attribution='&copy; <a href="http://osm.org/copyright">
        OpenStreetMap</a> contributors'
    url='http://{s}.tile.osm.org/{z}/{x}/{y}.png'
    />
    { this.state.geojsonUrl && this.state.geojson &&
      <GeoJSON
      key={this.state.geojsonUrl}
      data={this.state.geojson}
      style={this.featureStyle}
      pointToLayer={this.pointToLayer}
      onEachFeature={this.onEachFeature}
      >
      </GeoJSON>
    }
    </Map>
    </div>
  );
}
```

Listing 5.1: `render` method of the Map component of the geographical visualization

The `render` method is the only required method of a React component. It will be invoked on the initial rendering of the component of the `DOM` and on every update of the component's properties. React's templating language "`JSX`" allows to nest other child components in to the React parent component. In this case the `Map` component includes a `TileLayer GeoJSON` component from the `react-leaflet` library. This library conveniently provides "React components for Leaflet maps." [10].

5.3.1 GeoJSON component

The `GeoJSON` component is provided by the `Map` component with a couple of properties: It gets a (1) geojsonURL as well as a (2) geojson as data attribute. Furthermore a couple of callbacks is passed into the child component, including (3) featureStyle, (4) pointToLayer and (5) onEachFeature.

This way, the parent `Map` component controls the data flow and without a `geojson` object, no polygons are placed on the map. A changed `geojsonURL` will always update the child component as it is used a `key` on the `GeoJSON` component. The callbacks passed into the `GeoJSON` component control the visual representation of each polygon and they add event handlers for a mouse click or a mouse move on each polygon. Listing 5.2 shows the event handlers added to the map.

```
onEachFeature(feature: geojson.Feature<geojson.GeometryObject>,
     layer: Leaflet.Layer){
  this.state.layerList.push(layer);
  layer.on({
    mouseover: () => {
      this.state.controller.publish('mcv.select.highlight', [
          Number(feature.id)]);
    },
    click: (event: Leaflet.LeafletMouseEvent) => {
      if(event.originalEvent.ctrlKey){
        this.state.controller.publish('mcv.select.focus', this.
            xor(this.state.focusedIds, Number(feature.id)));
      } else {
        this.state.controller.publish('mcv.select.focus', [
            Number(feature.id)]);
      }
    }
  });
}
```

Listing 5.2: `onEachFeature` callback, adding handlers for mouse events

First we cache all layers in the internal state of the `Map` component. On each `mouseover` event, the `id` of the feature is published as `mcv.select.highlight` interaction. A `click` event is distinguised if the control key is pressed or not. In the latter case, the id of the feature is either added or removed from the list of focused ids and then the list of focused ids is published as `mcv.select.focus` interaction.

```
featureStyle(feature: geojson.Feature<geojson.GeometryObject>):
     Leaflet.PathOptions{
  const focused = (this.state.focusedIds.includes(Number(
      feature.id)))
  const fillColor = focused ?  Color('red') : Color('blue');
  let color = fillColor;
  let weight = 2;
  let dashArray = '3';
  if (this.state.highlightedIds.includes(Number(feature.id))) {
    weight = 4;
    color = 'white';
    dashArray = '';
  }
  return { color, weight, fillColor, dashArray };
}
```

Listing 5.3: `featureStyle` callback, configuring the visual appearance depending on the currently highlighted or focused feature ids

The `featureStyle` in Listing 5.3 method is very straightforward. If the feature is currently focused, the `fillColor` of the polygon is red, otherwise blue. Likewise, if the feature is currently highlighted, the polygon has a white, solid stroke.

```
pointToLayer(geoJsonPoint:any, latlng: Leaflet.LatLng){
  return new Leaflet.CircleMarker(latlng);
}
```

Listing 5.4: `pointToLayer` callback, if a feature of `GeoJSON` has a point geometry, it will be shown as a circle

Finally, we configure how to display point geometries in callback `poinToLayer`. Since normal markers do not have a configurable color and style, we instruct the `GeoJSON` component to render a `CircleMarker` for each point geometry instead. This way, the same options of `featureStyle` can be applied to both point and area geometries.

5.4 MultiviewController

The `MultiviewController` is a simple wrapper around the library `PubSubJS` and works as a broker for interactions across coordinated multiple views.

As the central part of the software, the `MultiviewController` is also used for certain performance optimiziations. These optimizations affect how often the geometries are reloaded, i.e. it reduces the number of requests to the `GeoJSON` endpoint.

```
import PubSub = require('pubsub-js');

export class MultiviewController {
  private _geojsonUrl: string;

  public subscribe(msg:string, callback: (msg:string, data:any)
      => void) {
    PubSub.subscribe(msg, callback);
  }

  public publish(msg:string, data:any) {
    if (msg === 'mcv.reconfigure.url'){
      if((this._geojsonUrl == null) || (this._geojsonUrl !==
          data)){
        this._geojsonUrl = data;
        fetch(data, {
          credentials: "same-origin"
        }).then((resp) => resp.json()).then((response) => {
          PubSub.publish('mcv.reconfigure.geometry', response);
          PubSub.publish(msg, data);
        }).catch((err) => {
          console.log(err);
        })
      }
    } else {
      PubSub.publish(msg, data);
    }
  }

  public clearAllSubscriptions() {
    PubSub.clearAllSubscriptions();
  }
}
```

Listing 5.5: The `multiviewController` avoids unnecessary requests to reload geometries

As shown in Listing list:implementation:multiviewController, the geometries depend on a change of the `url` of the geometries. A change of the `url` will automatically trigger an interaction of type `reconfigure` to reload geometries.

An example of these geometries can be seen in listing 5.6. For convenience, the file includes the aggregated data in the `properties` of each feature. In this case the colors the shapes of the 2.5D tree map are based on the value of `user_count_normalized`. Each feature comes with a unique `id` which is published when a user interacts with the feature.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "MultiPolygon",
        "coordinates": []
      },
      "properties": {
        "NAME_1": "Baden-Württemberg",
        "state_code": "BW",
        "user_count_total": "34",
        "user_count_normalized": "0.10149253731343283"
      },
      "id": 0
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": []
      },
      "properties": {
        "NAME_1": "Bayern",
        "state_code": "BY",
        "user_count_total": "36",
        "user_count_normalized": "0.10746268656716418"
      },
      "id": 1
    }
  ]
}
```

Listing 5.6: A truncated example of a user distribution across German federal states

We can use this data as input for our common visual analytics platform, e.g. figure 5.4 shows the user distribution of Rundfunk MITBESTIMMEN.

5.5 Implemented interactions

In the course of this thesis we want to implement the following interactions:

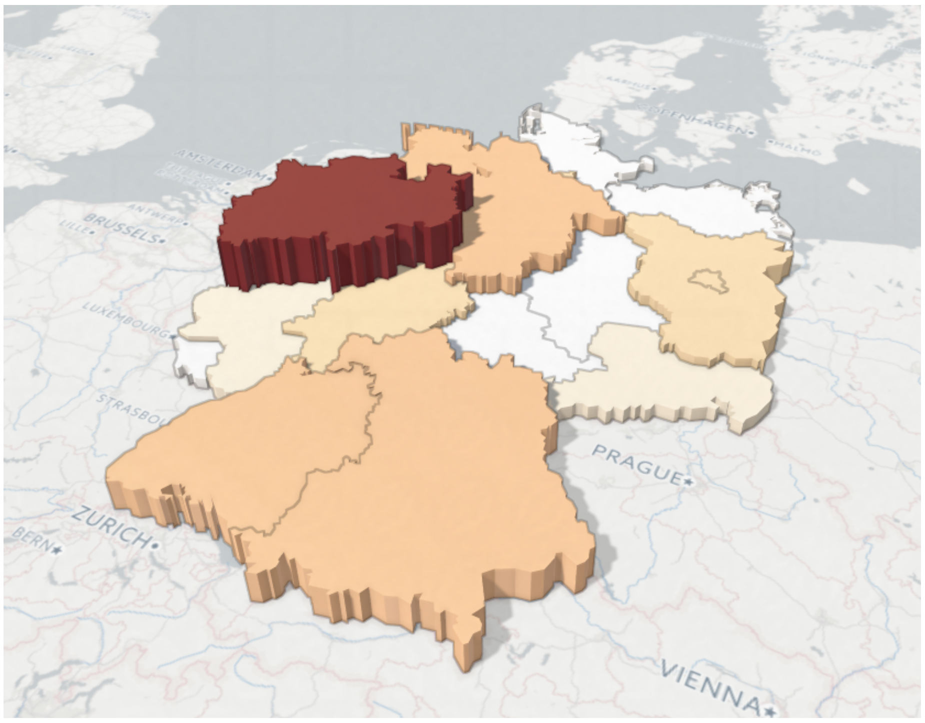


Fig. 5.4: User distribution of RUNDFUNK MITBESTIMMEN across German federal states

- *Select*: The user clicks on a building or region in a geographical map and all affected properties in the 2.5D tree map will be highlighted.
- *Explore*: The user clicks on a block in the 2.5D tree map and the viewpoint in the geographical map will be centered on relevant area.
- *Reconfigure*: The user selects a different feature set and the changes are reflected in both the geographical map (e.g. point instead of polygon geometries) and in the 2.5D tree map.
- *Filter*: The user double-clicks on a region in a geographical map and the 2.5D tree map will be based on data of only that region.

6

Evaluation

How and what can we evaluate?

The performance?

The flexibility?

6.1 Comparison of client-side component frameworks

This section evaluates the most suitable client-side rendering and component based JavaScript framework for coordinated multiple views and whether or not to use web components.

Three JavaScript frameworks have been evaluated: (1) GlimmerJS (2) Google Polymer and (3) ReactJS. Google Polymer is built on top of web components and GlimmerJS applications can be exported as a web component, but React does not support web components.

Web components is a recent standard of the W3C[2] to bring component-based software engineering to the world wide web. Web components are a set of web platform APIs to create new custom, reusable, encapsulated HTML tags that can be used in web pages and web applications [30]. If coordinated multiple views are implemented in JavaScript based web applications, web components are a promising choice, to allow arbitrary views to be put together.

GlimmerJS is the rendering enginge of EmberJS[43]. In 2017 it was released as a standalone component framework. Applications written in GlimmerJS can be exported as web components. These web components can be included in any website, which makes GlimmerJS a reasonable choice to build high-quality widgets for user interfaces. GlimmerJS also uses handlebars[26], a user-friendly templating language. The downside of GlimmerJS is the current lack of documentation and immaturity due to the recent first release this year.

Google Polymer is another popular library to build web components [24]. With 18,469 stars on Github it is the most popular framework for web components at the time of writing. Polymer has a large community and comprehensive documentation and therefore more suitable than GlimmerJS to build coordinated multiple views.

But critically, coordinated multiple views require a means to exchange data between views, which is specific to interactions. The web component specification, unfortunately, does not specify how arbitrary JavaScript objects can be passed to web components. String based attributes are supported, as seen in Listing 6.1. To pass rich data to components however, web component frameworks have to roll their own data flow and syntax. Google Polymer's syntax to pass rich data to a components is shown in Listing 6.2. But this is a proprietary solution that abandons standard HTML.

```
<google-map fit-to-marker api-key="AIzaSyD3E1D9b-
    Z7ekrT3tbhl_dy8DCXuIuDDRc">
  <google-map-marker latitude="37.78" longitude="-122.4"
      draggable="true"></google-map-marker>
</google-map>
```

Listing 6.1: An example of string based attributes of web components [6]

```
<some-component some-prop="{{richData}}"></some-component>
```

Listing 6.2: An small syntax example how Google Polymer passes rich data to a component

This raises some problems in existing applications: A particular component-based frontend framework can not be assumed, a lot of existing applications are also written without any JavaScript framework. Proprietary solutions like the one of Polymer lessen the main motivation of implementing against web components: Platform agnostic flexibility.

ReactJS is a JavaScript library for building user interfaces[23]. React is explicitly not implementing web components and is not going to implement web components in the future. It has, in return, a well-known way of integrating the component framework into a legacy application built with e.g. jQuery. Along with its major advantage of easy integration, it has a striving community, lots of documentation and tutorials and it is well tested.

As a summary, if there is (1) no obligation to implement web components (2) and an easy integration into an existing application is necessary, then React is the perfect choice for coordinated multiple views. Figure 6.1 shows the pros and cons of each framework for the use case.

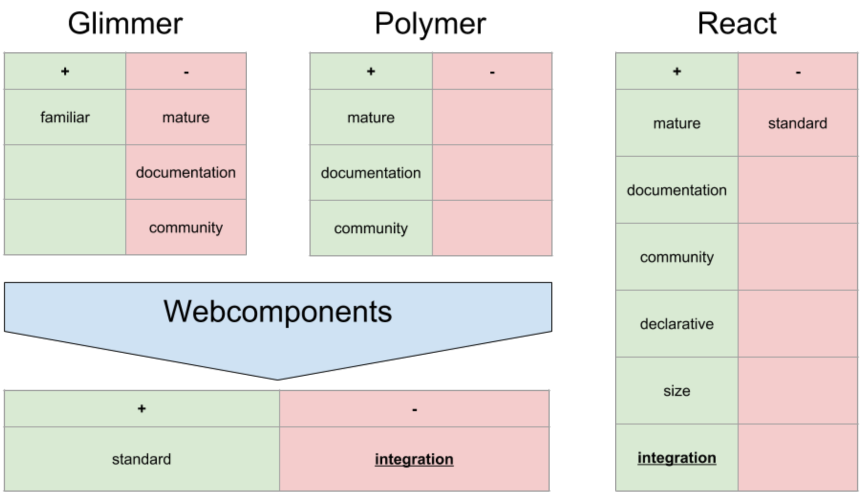


Fig. 6.1: Comparison of frontend frameworks

6.2 Use case

No section without text

6.2.1 Data Sets

For our uses cases, we have two different data sets: One data set consists of a user ranking of public broadcasting in Germany, i.e. entities, mostly TV and radio broadcasts, are liked or disliked by people. This data is public data and can be used by media researchers of broadcasting corporations but also targets media journalists and the general audience. The other set, called `RISO`, consists of statistical data from various German administrations and is used by the authorities for urban planning and policy strategies.

Both data sets share some characteristics. The administrative data connects certain features with certain regions of Germany. As Germany is a federal state, larger regions consist of many other smaller regions.

The second one consists of user data that was collected through a web application called RUNDFUNK MITBESTIMMEN.

RISO

The `RISO` data base is used in by local authorities to get insights about governmental KPIs to assist local and regional decision making. It is a relational database and a part of the data base schema is shown as an ER diagram in Figure 6.2.

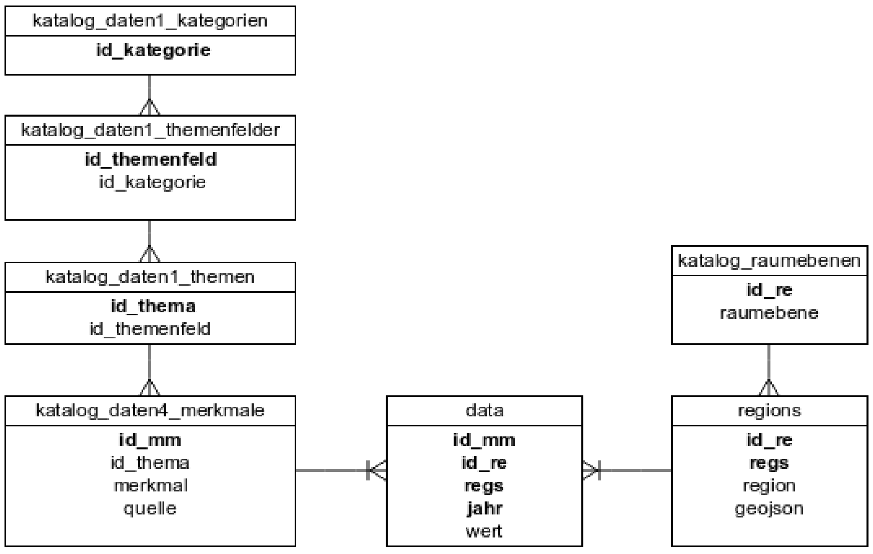


Fig. 6.2: Part of the `RISO` database schema. Primary keys are set in bold.

The largest table is called `data` with approximately 10,466,600 records, which holds all values along with the survey date. **Features** This data is connected to a feature table through a foreign key called `id_mm`. In the feature table we can find the description for every referenced feature, e.g. population density, working population in agriculture, education spending. The RISO system groups all features in a 4-level hierarchy:

1. `katalog_daten_1_kategorien`
2. `katalog_daten_2_themenfelder`
3. `katalog_daten_3_themen`
4. `katalog_daten_4_merkmale`

The actual features table is the last one in the list. At the lowest level within the hierarchy, this is the largest table with 1234 records.

Regions On the other side, the geographical data is stored in the `regions` table. The geometry data for each region is stored in the `geojson` column and as the name suggests, the data type is a `geojson`. The foreign keys that connect the tables `data` and `regions` are called `id_re` and `regs`. Unlike the feature table, the regions are grouped through the `id_re` that indicates the hierarchy level. So the values of the `id_re` column denominate the level of the hierarchy. E.g. a region with a `id_re` of 1 is a federal state of Germany, a region with id 13 is a constituency. A textual description for the hierarchy level can be found in the `katalog_raumebenen` table in column `raumebene`. Both column `id_re` and `regs` belong to the primary key of the regions table, so there will never be two regions on the same hierarchy level with the same `regs` id.

Characteristics As we can see, the schema of the RISO database follows a rather denormalized approach. The schema does not make a lot of assumptions regarding the input data. It allows to add data of arbitrary size, features and completeness as long as there is some kind of numerical data associated with some kind of geographical unit. This approach is suitable for a data base that incorporates data from different sources, as it is the case with the RISO data base.

Rundfunk MITBESTIMMEN

Unlike the RISO database, the data base of Rundfunk MITBESTIMMEN is used as persistence layer. For that reason the data base schema follows the requirements of a web application in production.

As outlined in Section 1.1 Rundfunk MITBESTIMMEN is an evaluation platform for public broadcasting in Germany. First, users vote on broadcasts, i.e. they decide if they want to support broadcast or if they do not want to support. As a next step, user can define a weighting by distributing a virtual, monthly budget among the chosen broadcasts.

Figure 6.3 shows the data base schema of the application. A `user` is connected to `broadcast` through a `selection`. If the `user` supports some a broadcast, the `response` on the given `selection` will be 'positive'. If the `user` does not wish to support a `broadcast`, the `response` will be 'neutral'. The `user` can allocate virtual money to supported broadcasts. The money will be stored in the column `amount` of the `selection`. The sum of all amounts for one user will never exceed the virtual budget of 17.50€.

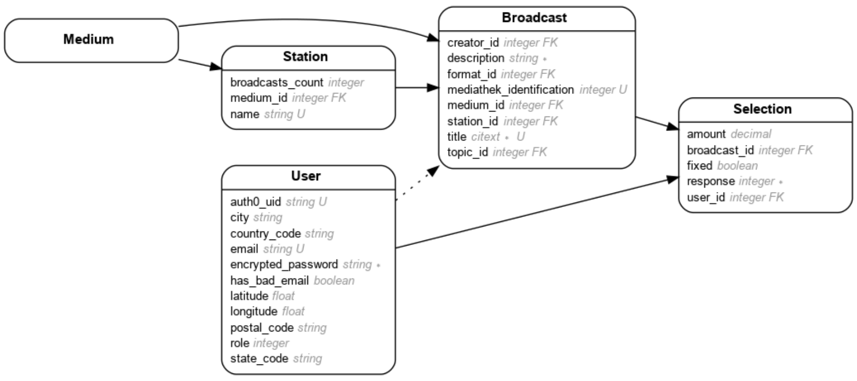


Fig. 6.3: Database schema of the Rundfunk MITBESTIMMEN application

Features We have both numerical as well as nominal features. A numerical feature could be the number of supporters from an area in Germany. A nominal feature could be a list of the most supported broadcasts from an area in Germany. Numerical and nominal features can be combined, so we could request for every region, a distribution of the desired expenditure for radio, TV, online and other broadcasts.

Regions Rundfunk MITBESTIMMEN stores the geometry for each region in `geojson` files. These files hold a `FeatureCollection`. Every `Feature` is a region, the identifier is stored as a property. We merge the geometry data with features for every request. To be precise: We get all the user data, group it by the identifier `state_code` and merge it with the geometry in the `geojson`.

Characteristics The data base schema is a result of the specific requirements of the persistence layer. Changes in the source code may require a migration of the data base schema.

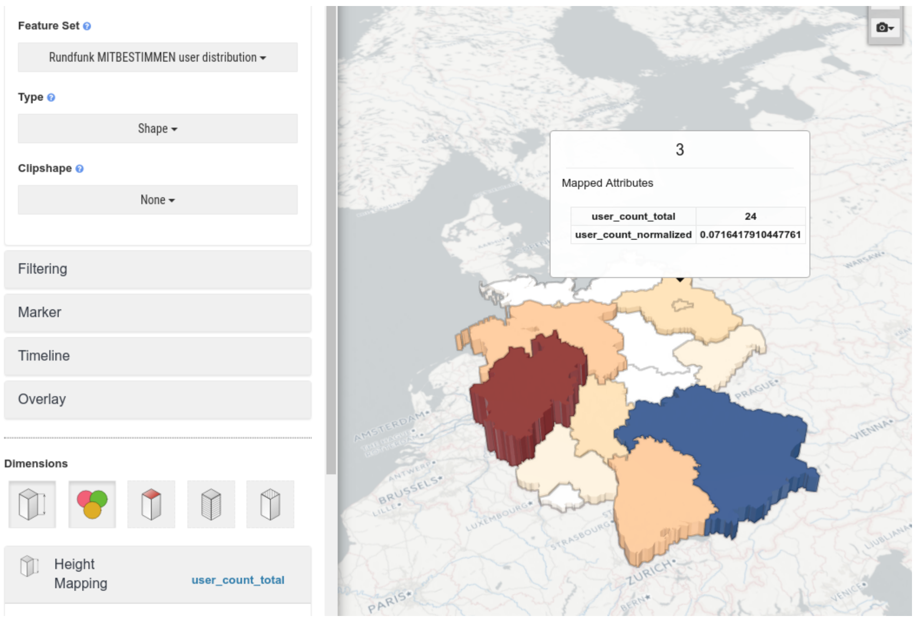
However, we can ask a lot of questions already with common data base queries or standard data analysis tools:

1. How does the actual support of a broadcast compare to the average support of a broadcast?
2. What are the most popular broadcasts in Berlin?
3. What is the desired ratio of genres of supported broadcasts? How important is education compared to sport?
4. How does the support of a broadcast change over time?
5. According to the user ranking, which broadcasts are similar to each other?

6.3 Existing Interactions

We will now classify the implemented interactions of our two applications according to the classification by Yi, Kang, and Stasko [44] in Section 2.2.1.

In our visual analytics platform possible interactions can be categorized into the classes *select*, *explore*, *reconfigure*, *encode* and *filter*. As seen in Figure 6.4 the user can *select* one item in the view by clicking on it. The user can reveal a tooltip showing the item properties by hovering with the mouse on the item, which is another *selection*. The user can *explore* the map in the usual manner: If the user drags with the mouse on the map, a panning operation is performed with the viewpoint focused on Germany, i.e. the camera moves around like a turntable. The zoom factor can be changed by scrolling on the canvas of the map. *Encode* and *reconfigure* techniques are performed through the menu on the left side: Under the “features” tab, the user can *reconfigure* different data sets and the displayed diagram, e.g. a tree map visualization based on the geometry shape, cubes or voronoi regions. The tab “Dimensions” allows the user to *encode* properties of a data set to visual attributes, e.g. the height, color and texture of an item. The tab “Filter” can be used to reduce the displayed data set along a range of continuous values. Figure 6.5 shows the range of visible values in the left menu. When the user drags the slider, the items in the map on the right side are updated interactively.



Feature Set

Rundfunk MITBESTIMMEN user distribution

Type

Shape

Clipshape

None

Filtering

Marker

Timeline

Overlay

Dimensions

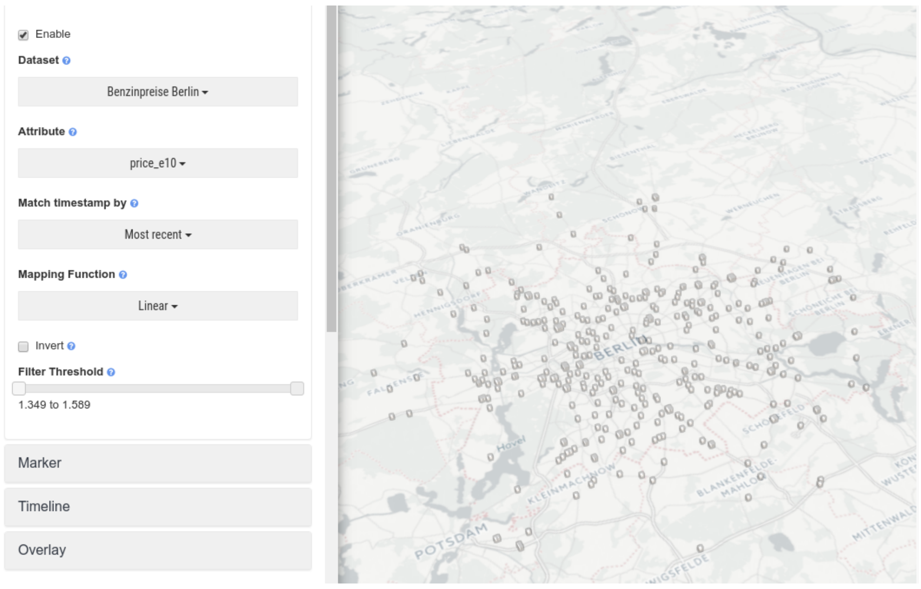
Height Mapping user_count_total

3

Mapped Attributes

user_count_total	24
user_count_normalized	0.0716417910447761

Fig. 6.4: Items can be highlighted with a click, Bavaria is currently highlighted. A mouse over reveals a tooltip showing item properties. The menu on the left side allows to change the data set and the specific base visualization.



☑ Enable

Dataset

Benzinpreise Berlin

Attribute

price_e10

Match timestamp by

Most recent

Mapping Function

Linear

☐ Invert

Filter Threshold

1.349 to 1.589

Marker

Timeline

Overlay

Fig. 6.5: Only gas stations with a price for E10 within 1.349 Euro and 1.589 Euro are displayed in the map

6.4 Planned Interactions

We have a focus on coordinated multiple views consisting of a tree map and a geographical map. Let's have some examples how an interaction between a 2.5D tree map and a 2D map might work:

1. User selects a feature set from the drop down in the menu. This will trigger a *Reconfigure* interaction. A data set consisting of all features and their ids, geometries and metadata is transferred. The receiving components are both the 2.5D tree map and the 2D map which will rerender the entire visualization.
2. User hovers with a mouse over a polygon in the 2D map. This will trigger a *Select* interaction. The data is a single feature id that will be transferred to the 2.5D tree map, which will change the color of an box.
3. Rotate or zoom the 2.5D tree map. This will also rotate or zoom the 2D map. The interaction would fall into *Explore* and the shared information is the orientation of the camera and the zoom level.
4. A click in the 2.5D tree map will trigger an *Explore* interaction. The data is a single feature id sent to the 2D map. The map will center the viewport on the center of geometry of the respective feature.
5. The user selects many features at once in the 2D map by dragging a rectangle. The ids of all features within the rectangle are sent to the 2.5D tree map. All features will be highlighted with a different color, which is therefore a *Select* interaction.
6. *Reconfigure* the layouting of the 2.5D tree map by choosing a different hierarchy level. This increased granularity may lead to an increased granularity in the 2D map, e.g. show postal codes instead of federal states. The changed data are additional items, that are nested in the former items.
7. *Encode* the 2.5D tree map by a different attribute mapping like color, height or texture. If the 2D map has no geometry data that defines the shape of a feature, it can also display a larger point marker.
8. Apply a *Filter* and reduce the data set by choosing only items with metadata beyond a certain treshold. The reduced data leads to a full re-render of all data visualizations. The message contains the updated item list
9. Show a *Connect* by highlighting boxes of the same subtree in the 2.5D tree map. The respective connected items would be highlighted in the 2D map as well. Here the data is a relation between items.

What are the key interactions in our use case?

7 Conclusion and Future Work

Did we achieve our goals?

Is the concept sane in regarding the implementation?

List stuff which was not accomplished in this master thesis

References

[1] Internet Engineering Task Force (IETF). *GeoJSON*. Aug. 1, 2016. URL: `http://geojson.org/` (visited on 11/20/2017).

[2] World Wide Web Consortium (W3C). *Web components current status*. Feb. 13, 2017. URL: `https://www.w3.org/standards/techs/components` (visited on 07/24/2017).

[3] Vladimir Agafonkin and OpenStreetMap contributors. *Leaflet*. Aug. 1, 2017. URL: `http://leafletjs.com/` (visited on 11/08/2017).

[4] Benjamin Bach et al. "A Review of Temporal Data Visualizations Based on Space-Time Cube Operations". In: *Eurographics Conference on Visualization (EuroVis 2014)*. 2014, pp. 23–41.

[5] Jacques Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. Jan. 2010. ISBN: 978-1-58948-261-6.

[6] Eric Bidelman. *google-map*. Nov. 18, 2017. URL: `https://www.webcomponents.org/element/GoogleWebComponents/google-map` (visited on 11/20/2017).

[7] Staffan Bjork, Lars Erik Holmquist, and Johan Redstrom. "A framework for focus+ context visualization". In: *Information Visualization, 1999.(Info Vis' 99) Proceedings. 1999 IEEE Symposium on*. IEEE. 1999, pp. 53–56.

[8] Thomas Bladh, David A. Carr, and Jeremiah Scholl. "Extending Tree-Maps to Three Dimensions: A Comparative Study". In: *Computer Human Interaction: 6th Asia Pacific Conference, APCHI 2004, Rotorua, New Zealand, June 29-July 2, 2004. Proceedings*. Ed. by Masood Masoodian, Steve Jones, and Bill Rogers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 50–59. ISBN: 978-3-540-27795-8. DOI: `10.1007/978-3-540-27795-8_6`. URL: `http://dx.doi.org/10.1007/978-3-540-27795-8_6`.

[9] Mike Bostock. *Crossfilter*. July 2017. URL: `http://square.github.io/crossfilter/` (visited on 07/10/2017).

[10] Paul Le Cam. *react-leaflet*. Nov. 10, 2017. URL: `https://github.com/PaulLeCam/react-leaflet/commits/master` (visited on 11/17/2017).

[11] MST Carpendale. “Considering visual variables as a basis for information visualisation”. In: (2003).

[12] John Corbett. “Charles Joseph Minard, Mapping Napoleon’s March, 1861.” In: (2001).

[13] Alan Dix and Geoffrey Ellis. “Starting Simple: Adding Value to Static Visualisation Through Simple Interaction”. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI ’98. L’Aquila, Italy: ACM, 1998, pp. 124–134. DOI: `10.1145/948496.948514`. URL: `http://doi.acm.org/10.1145/948496.948514`.

[14] Jürgen Döllner. *Visualization Techniques for Big Data*. July 2017. URL: `https://hpi.de/doellner/masterarbeiten/visual-software-analytics.html` (visited on 07/11/2017).

[15] Patrick Th. Eugster et al. “The Many Faces of Publish/Subscribe”. In: *ACM Comput. Surv.* 35.2 (June 2003), pp. 114–131. ISSN: 0360-0300. DOI: `10.1145/857076.857078`. URL: `http://doi.acm.org/10.1145/857076.857078`.

[16] Stephen Few. “Data visualization for human perception”. In: *The Encyclopedia of Human-Computer Interaction, 2nd Ed.* (2013).

[17] Pablo Figueroa, Mark Green, and Benjamin Watson. “A framework for 3D interaction techniques”. In: *CAD/Graphics*. Vol. 8. 2001, pp. 22–24.

[18] Michael Friendly and Daniel J Denis. “Milestones in the history of thematic cartography, statistical graphics, and data visualization”. In: *URL http://www. datavis. ca/milestones* 32 (2001).

[19] Simone Garlandini and Sara Fabrikant. “Evaluating the effectiveness and efficiency of visual variables for geographic information visualization”. In: *Spatial information theory* (2009), pp. 195–211.

[20] Macro connection group. *The Observatory of Economic Complexity*. July 2017. URL: `http://atlas.media.mit.edu/en/profile/country/deu/` (visited on 07/03/2017).

[21] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. “A tour through the visualization zoo”. In: *Queue* 8.5 (2010), p. 20.

[22] Quan Ho. *Architecture and Applications of a Geovisual Analytics Framework*. May 2013, p. 78. ISBN: 978-91-7519-643-5.

[23] Facebook Inc. *React, A javascript library for building user interfaces*. Sept. 7, 2017. URL: `https://facebook.github.io/react/` (visited on 09/23/2017).

[24] Google Inc. *Polymer Project*. Sept. 22, 2017. URL: `https://www.polymer-project.org/` (visited on 09/23/2017).

[25] Brian Johnson and Ben Shneiderman. "Tree-Maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures". In: *Proceedings of the 2Nd Conference on Visualization '91*. VIS '91. San Diego, California: IEEE Computer Society Press, 1991, pp. 284–291. ISBN: 0-8186-2245-8. URL: `http://dl.acm.org/citation.cfm?id=949607.949654`.

[26] Yehuda Katz. *handlebars*. June 22, 2017. URL: `http://handlebarsjs.com/` (visited on 09/23/2017).

[27] Daniel A. Keim. "Information Visualization and Visual Data Mining". In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (Jan. 2002), pp. 1–8. ISSN: 1077-2626. DOI: `10.1109/2945.981847`. URL: `http://dx.doi.org/10.1109/2945.981847`.

[28] Sam Kusinitz. *12 Reasons to Integrate Visual Content Into Your Marketing Campaigns*. July 18, 2014. URL: `https://blog.hubspot.com/marketing/visual-content-marketing-infographic` (visited on 07/08/2017).

[29] Nada Lavrač et al. "Data mining and visualization for decision support and modeling of public health-care resources". In: *Journal of Biomedical Informatics* 40.4 (2007). Public Health Informatics, pp. 438–447. ISSN: 1532-0464. DOI: `http://dx.doi.org/10.1016/j.jbi.2006.10.003`. URL: `http://www.sciencedirect.com/science/article/pii/S153204640600116X`.

[30] Sam Li, Douglas Stockwell, and Andy Mutton. *What are web components?* Nov. 17, 2017. URL: `https://www.webcomponents.org/introduction` (visited on 11/20/2017).

[31] Daniel Limberger et al. "Dynamic 2.5D Treemaps Using Declarative 3D on the Web". In: *Proceedings of the 21st International Conference on Web3D Technology*. Web3D '16. Anaheim, California: ACM, 2016, pp. 33–36. ISBN: 978-1-4503-4428-9. DOI: `10.1145/2945292.2945313`. URL: `http://doi.acm.org/10.1145/2945292.2945313`.

[32] Richard E. Mayer. "Multimedia learning: Are we asking the right questions?" In: *Educational Psychologist* 32.1 (1997), pp. 1–19. DOI: `10.1207/s15326985ep3201\_1`. eprint: `http://dx.doi.org/10.1207/s15326985ep3201_1`. URL: `http://dx.doi.org/10.1207/s15326985ep3201_1`.

[33] Andrew McAfee and Erik Brynjolfsson. *Big Data: The Management Revolution*. Oct. 2012. URL: `https://hbr.org/2012/10/big-data-the-management-revolution` (visited on 07/08/2017).

[34] Thomas Nocke and Heidrun Schumann. “Meta Data for Visual Data Mining”. In: *Proceedings Computer Graphics and Imaging, CGIM’02*. Citeseer. 2002.

[35] Thiago Poleto, Victor Diogho Heuer de Carvalho, and Ana Paula Cabral Seixas Costa. “The Roles of Big Data in the Decision-Support Process: An Empirical Investigation”. In: *Decision Support Systems V – Big Data Analytics for Decision Making: First International Conference, ICDSST 2015, Belgrade, Serbia, May 27-29, 2015, Proceedings*. Ed. by Boris Delibašić et al. Cham: Springer International Publishing, 2015, pp. 10–21. ISBN: 978-3-319-18533-0. DOI: `10.1007/978-3-319-18533-0_2`. URL: `http://dx.doi.org/10.1007/978-3-319-18533-0_2`.

[36] Severino Ribecca. *The Data Visualisation Catalogue*. Sept. 11, 2017. URL: `http://www.datavizcatalogue.com/index.html` (visited on 09/23/2017).

[37] J. C. Roberts. “State of the Art: Coordinated Multiple Views in Exploratory Visualization”. In: *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007)*. July 2007, pp. 61–71. DOI: `10.1109/CMV.2007.20`.

[38] Morgan Roderick. *PubSubJS*. May 21, 2017. URL: `https://github.com/mroderick/PubSubJS` (visited on 11/09/2017).

[39] Ben Shneiderman. “The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations”. In: *Proceedings of the 1996 IEEE Symposium on Visual Languages*. VL ’96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 336–. ISBN: 0-8186-7508-X. URL: `http://dl.acm.org/citation.cfm?id=832277.834354`.

[40] Ben Shneiderman. “Tree Visualization with Tree-maps: 2-d Space-filling Approach”. In: *ACM Trans. Graph.* 11.1 (Jan. 1992), pp. 92–99. ISSN: 0730-0301. DOI: `10.1145/102377.115768`. URL: `http://doi.acm.org/10.1145/102377.115768`.

[41] Frank Treacy. *Getting Started with Ember and Data Down Actions Up*. 2017. URL: `https://emberigniter.com/getting-started-ember-cli-data-down-actions-up-tutorial/` (visited on 07/23/2017).

[42] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. “Guidelines for Using Multiple Views in Information Visualization”. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI ’00. Palermo, Italy: ACM, 2000, pp. 110–119. ISBN: 1-58113-252-2. DOI: `10.1145/345513.345271`. URL: `http://doi.acm.org/10.1145/345513.345271`.

[43] Wikipedia. *Ember.js*. Sept. 20, 2017. URL: `https://en.wikipedia.org/wiki/Ember.js` (visited on 09/23/2017).

[44] J. S. Yi, Y. a. Kang, and J. Stasko. "Toward a Deeper Understanding of the Role of Interaction in Information Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1224–1231. ISSN: 1077-2626. DOI: `10.1109/TVCG.2007.70515`.