Multiple Coordinated Views on Massive Geo Data

Master's Thesis in partial fulfillment for the academic degree

"Master of Science" (M.Sc.)

in IT Systems Engineering

Hasso Plattner Institute | Faculty of Digital Engineering
University of Potsdam

submitted by

Robert Schäfer

robert.schaefer@student.hpi.de

Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany

Supervisors:

Prof. Dr. Jürgen Döllner
Dr. Benjamin Hagedorn
Jan Klimke

Hasso Plattner Institute
Potsdam, Germany
December 14, 2017

Abstract

Coordinated, multiple information visualizations can offer added value by combining the strengths of individual visualization techniques or data representations and by reflecting user interactions among multiple views simultaneously. Specifically, when visualizing multi-dimensional and hierarchical geodata, the combination of digital maps and treemap-based visualizations can facilitate the comprehensibility and interactivity. E.g. a linked geographic map can establish the geographical context if the used treemap layout algorithm does not take into account the spatial data. In addition, the user can select elements as desired based on the geographic proximity in the map or based on the technical context in the treemap.

This master thesis provides the design and implementation of a coordinated visualization of a treemap and a geographic visualization. The focus is on coordinated interactions, the information which is exchanged between individual views and the common intersection between data visualizations.

Based on existing scientific work and a series of plausible interactions in individual data visualizations, a conceptual framework for coordinated multiple views is specified. Furthermore, a formal concept of an interaction is defined, a common data model between visualizations is derived and a notification procedure is developed. This specification is implemented for the use case of the coordination of a treemap and a geographic visualization. Finally, common application scenarios and software-relevant requirements are evaluated and the system performance is analysed.

Zusammenfassung

Koordinierte, multiple Informationsvisualisierungen können einen Mehrwert bieten, indem sie die Stärken einzelner Visualisierungstechniken oder Datendarstellungen kombinieren und Benutzerinteraktionen gleichzeitig in mehreren Ansichten wiedergeben. Insbesondere bei der Visualisierung mehrdimensionaler und hierarchischer Geodaten kann die Kombination von digitalen Karten und Treemap-basierten Visualisierungen die Verständlichkeit und Interaktivität erleichtern. Z.B. kann eine verknüpfte Kartendarstellung den geografischen Kontext herstellen, wenn der verwendete Layoutalgorithmus der Treemap die räumlichen Daten nicht berücksichtigt. Darüber hinaus kann der Nutzer je nach Bedarf Elemente basierend auf der geograpischen Nähe in der Karte oder basierend auf dem technischen Kontext in der Treemap auswählen.

Diese Masterarbeit beinhaltet den Entwurf und zeigt die Implementierung einer koordinierten Visualisierung einer Treemap und einer geographischen Visualisierung. Der Fokus liegt auf koordinierten Interaktionen, den Informationen, die dabei zwischen einzelnen Ansichten ausgetauscht werden, und den gemeinsamen Schnittmengen zwischen einzelnen Datenvisualisierungen.

Aufbauend auf bestehenden wissenschaftlichen Arbeiten und einer Reihe von plausiblen Interaktionen in eigenständigen Datenvisualisierungen wird ein konzeptueller Rahmen spezifiziert. Dabei wird ein formales Konzept einer Interaktion definiert, ein gemeinsames Datenmodell zwischen Visualisierungen abgeleitet und ein Benachrichtigungsverfahren entwickelt. Diese Spezifikation wird für den Anwendungsfall der Koordination einer Treemap und einer geographischen Visualisierung implementiert. Anschließend werden typische Anwendungsszenarien und software-relevante Anforderungen evaluiert und eine Performance-Analyse durchgeführt.

Glossary

BMI Body mass index. 8

CMV Coordinated Multiple View. vii, 12, 20, 21

DOM Document Object Model. viii, 45, 49, 50

JSON JavaScript Object Notation. 61

Contents

Glossary v

1 Introduction 1
1.1 Motivation 1
1.2 Problem Statement 2
1.3 Hypothesis 3
1.4 Contributions 3
1.5 Structure of the Work 3

2 Related Work and Foundations 5
2.1 Information visualization 5
2.2 Data-driven Decision Support Systems 6
2.3 Geovisualization 6
2.3.1 Visual Variables 6
2.3.2 Choropleth maps 7
2.4 Information Visualization of Hierarchic Data 8
2.4.1 Treemaps 9
2.4.2 3D treemaps and 2.5D treemaps 10
2.5 Coordinated Multiple Views 11
2.5.1 Brushing and Linking 11
2.6 Technologies 13
2.7 Interaction Theory 14
2.8 Interaction Categories 14
2.9 Formalization of Interactions 15

3 Analysis 19
3.1 Existing Interactions 19
3.2 Software Requirements for Coordinated Multiple View (CMV)s 20
3.3 Component pattern and Web Frameworks 21

3.4 Arbitrary Data Visualization Techniques ... 23
3.4.1 Line Graphs ... 23
3.4.2 Bar Charts and Multi-Set Bar Charts ... 24
3.4.3 Histograms ... 25
3.4.4 Bubble Charts and Scatter Plots ... 25
3.4.5 Stacked Bar Charts ... 26
3.4.6 Hierarchical visualizations ... 27
3.4.7 Geographic Data Visualizations ... 29

4 Conceptual Framework ... 33
4.1 The Conceptual Framework from the User Perspective ... 33
4.1.1 Interaction ... 35
4.1.2 Interaction in Coordinated Multiple Views ... 35
4.1.3 View ... 35
4.1.4 Trigger ... 35
4.1.5 Effect ... 36
4.1.6 Category ... 36
4.1.7 Purpose ... 36
4.1.8 Subject ... 36
4.2 Shared data model ... 37
4.3 Encoding of Interaction Subjects ... 37

5 Implementation ... 43
5.1 Implemented interactions ... 43
5.2 Coordinated Multiple View Layout ... 44
5.3 Architecture ... 44
5.3.1 Update mechanisms ... 45
5.4 Notifications ... 45
5.4.1 Encoding of interaction category and purpose ... 46
5.5 MultiviewCoordinator ... 48
5.6 Render views in the Document Object Model (DOM) ... 49
5.7 Map Component ... 50
5.7.1 GeoJSON Component ... 51

6 Evaluation and Discussion ... 53
6.1 Use Cases and Discussion ... 53
6.1.1 Explain outliers ... 53
6.1.2 Multi-Select in 2.5D treemap ... 54
6.1.3 Bounding Box Selection in Geographic visualization ... 55
6.1.4 Use case summary ... 57
6.2 Performance Evaluation and Limitations ... 57

6.2.1 Immoscout 58
6.2.2 Landkreise 60
6.3 Evaluation of Requirements 60

7 Summary and Conclusion 63
7.1 Future Work 64

1 Introduction

The human brain processes visual information better than it processes text. As a result, data scientists often communicate their results with data visualizations. These visualizations make the data accessible to readers and help them to spot anomalies.

Data visualizations on a computer allow the user to interact with the data and explore different levels of granularity. Users can use an input device like a mouse or a keyboard to trigger an interaction and the visual representation of the data will change accordingly. In many cases a great interactivity results in a great user experience, as the user can change the visual representation as desired.

There is a wide range of existing frameworks and implementations for data visualizations. These frameworks often support interactions with additional controls and event handlers like mouse click events.

A few specific use cases extend these frameworks to show multiple views next to each other, with controls and event handlers to manipulate multiple views. The user experience is even better compared with single data visualizations but the development effort for multiple views is high. Currently, there is no dedicated or prevalent framework that helps to implement interactions in multiple visualizations of the same data.

1.1 Motivation

Many research papers in the field of coordinated multiple views have focused more on visual representations than interaction aspects. Even though interaction aspects give a great user experience and enable the user to find complex connections involving multiple views. Ho (2013) assumes this may "originate from the fact that the implementation of interaction techniques and interactive features normally takes much more time than the implementation of visual representations”. Obviously, there is a lot of existing research in the area but a lack of research regarding interactions in particular. This could also explain why hardly any general-purpose implementation exists to coordinate multiple views, that focuses on interactions. It is for this reason that this thesis develops an interaction model for coordinated multiple views, so future implementations can use this model as a specification.

Another motivation to do research in that particular area is a recent development in web application frameworks: Many popular frameworks like Angular, Ember, React and Vue have developed mechanisms to update UI elements during user interactions. These patterns and mechanisms have become so widespread and prevalent that they triggered even a web specification called “web components”. Obviously, these update mechanisms are a promising choice for coordinated multiple views. They have not gained research interest yet, probably because of the very recent development.

1.2 Problem Statement

Problem Statement
A 2.5D treemap of geographic data may loose the geographic context if the tiling algorithm is based on non-geographic attributes.

Let’s take the visualization of administrative districts in Germany as an example. An district that is located in the east of a second district may be placed in the 2.5D treemap left of it instead of being placed on the right side. If the tiling algorithm uses a non-geographic hierarchy, the geographic context will be lost entirely. Items that should belong together according to their geographic circumstances may be scattered across the 2.5D treemap. Users have a hard time to recognize geographic areas and locations which deteriorates the comprehensibility of the 2.5D treemap. Selecting and grouping items based on their geographic proximity becomes increasingly difficult if the items are scattered across the 2.5D treemap.

1.3 Hypothesis

Hypothesis
A second, geographic visualization next to the 2.5D treemap can preserve the geographic context if these two views are combined in a coordinated multiple view.

The user can relate an item in the 2.5D treemap if the item is linked with the corresponding item in the geographic visualization and vice versa. Many items can be selected in the geographic map based on their proximity by dragging a bounding box around them. In the 2.5D treemap the user can select many items based on their proximity in a non-geographic dimension and see the selection in the geographic visualization. Essentially, the limitations of a single 2.5D treemap or a single geographic visualization can be overcome by splitting up the interaction: The user can trigger the interaction in one view and see the effect, i.e. the change of visual representation, in another view.

1.4 Contributions

The contributions of this work are the following:

Contributions:
1. A formalization of interaction aspects in the field of data visualizations
2. A conceptual framework for coordinated multiple views of arbitrary data visualizations
3. An implementation of the conceptual framework for the present use case of geographic and hierarchic data
4. A proof of concept how 2.5D treemap and geographic visualization can be combined to overcome limitations of each visualization respectively

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. Each interaction is further examined to identify the relevant information that need to be communicated in a coordinated multiple view. The gained knowledge from that section is used in the following Section 4 to develop a conceptual framework that can be used for future implementations of coordinated multiple views. This conceptual framework includes a data model shared between all views, a suggested communication protocol as well as a formalization of an interaction. In Section 5 we describe the implementation of the conceptual framework for the present use case. This implementation serves as reference implementation, to prove feasibility of the conceptual framework. The implementation includes the necessary interactions to demonstrate the advantage of coordinated multiple views for hierarchic and geographic data. It also serves to validate or invalidate the hypothesis in Section 1.3. In Section 6 the implementation is tested for typical visual analytics tasks and it is demonstrated that additional value can be generated from the combination of 2.5D treemap and geographic visualization. A performance analysis of the implementation is carried out to demonstrate the feasibility of the conceptual framework.

Check design criteria or rephrase/delete this sentence

Based on the test data sets, the coordinated multiple view implementation is examined and evaluated for design criteria (Wang Baldonado, Woodruff, and Kuchinsky 2000) and general usability aspects (Roberts 2007). The types of evaluation are therefore:

(1) Use case, (2) Performance profiling and (3) manual check of formal requirements.

Finally, the main contributions in Section 7 are summarized and the future work is outlined.

2

Related Work and Foundations

This Chapter covers terminology, relevant techniques for data visualization and the related work in this area of research. Advantages and disadvantages of certain data visualization are emphasized. After introducing visualization techniques, related work on interaction aspects in coordinated multiple views is outlined.

2.1 Information visualization

Information visualization is a means of visual communication and have steadily developed since the 16th century (Friendly and Denis 2001). It is a generic term, expressing all kinds of effort to put data into visual context to help people understand the significance of data (Rose 2017). Information visualization today goes beyond standard charts and graphs used in spreadsheet applications and covers also infographics, heat maps, geographic visualizations and treemaps. Otherwise abstract information is visually represented, making complex data more accessible, understandable and usable.

Kusinitz (2014) mentions that the human brain processes visual information 60,000 times faster than text and visual content makes up even 93% of all human communication. According to the Interaction Design Foundation the purpose of data visualizations is twofold: 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." (Few 2013) Successful data visualizations helps the human user to derive knowledge and meta data from the visualization itself. Nocke and Schumann (2002) call this "visual data mining".

2.2 Data-driven Decision Support Systems

Data-driven decision support systems are applications to support businesses and organizational decision-making activities in which data visualizations play a key part (Lavrač et al. 2007) (Poleto, Carvalho, and Costa 2015). A common expression by impatient managers who can not afford to wade through lengthy reports has even become the title of a book about decision support systems: Stephen Few's "Show me the numbers".

In the business context, sales managers demand a quick access on the latest data with all relevant visualizations at once. Examples of this kind of decision support systems are called e.g. "decision cockpit" or "business sphere" (Davenport 2013).

We can expect to see these technologies more in more in business applications. McAfee and Brynjolfsson (2012) 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. In 1997 Mayer (1997) 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%.

Apparently, the application of combined data visualization techniques in decision making is a promising strategy.

2.3 Geovisualization

The umbrella term "Geovisualization" covers visualization techniques for the "visual exploration, analysis, synthesis and presentation of geospatial data (any data having geospatial referencing)" (MacEachren and Kraak 2001). The entities of geospatial data may include buildings, streets, landmasses, terrain, administrative districts and also moving entities like cars.

2.3.1 Visual Variables

French cartographer Jaques Bertin introduced seven visual variables in 1967 (Bertin 2010). 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 (2003) explains in detail their use in computational information instead of printed cartography. Garlandini and Fabrikant (2009) put these visual variables under systematic validation procedures. The authors conclude that the variable `size` provides the most accurate and efficient performance while the variable `orientation` provides the least performance. Bertin's visual variable play a role in interaction aspects of coordinated multiple views as they are used to communicate the effect of an interaction. A highlighted data point can be highlighted by changing the colour of an area or increasing the size of a point.

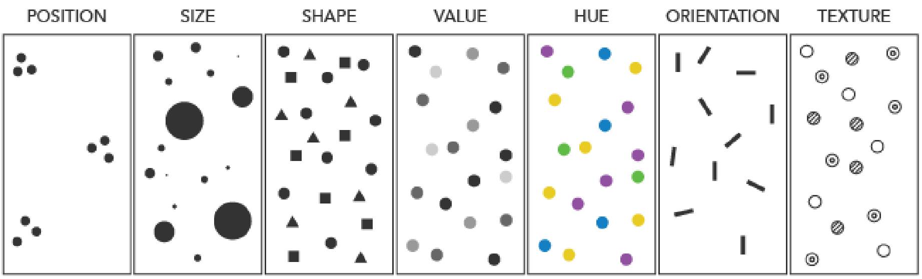


Fig. 2.1: Bertin's original visual variables (Foster 2017).

2.3.2 Choropleth maps

Choropleth maps are thematic maps in which areas are shaded or patterned in proportion to the statistical variables 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 2.2, showing the percentage of obese population in the US. Choropleth maps are very popular and therefore many people are familiar with them already. A downside of choropleth maps is that larger regions may appear more emphasized than smaller ones, since the entire area of regions is coloured. Another disadvantage of choropleth maps is the common error of incorrect encoding: Such an incorrect encoding would be the display of absolute numbers, e.g. total population or proceeds of crime, rather than relative numbers, e.g. population density or unemployment rate.

This work uses a choropleth map for the geographic visualization because of its widespread recognition.

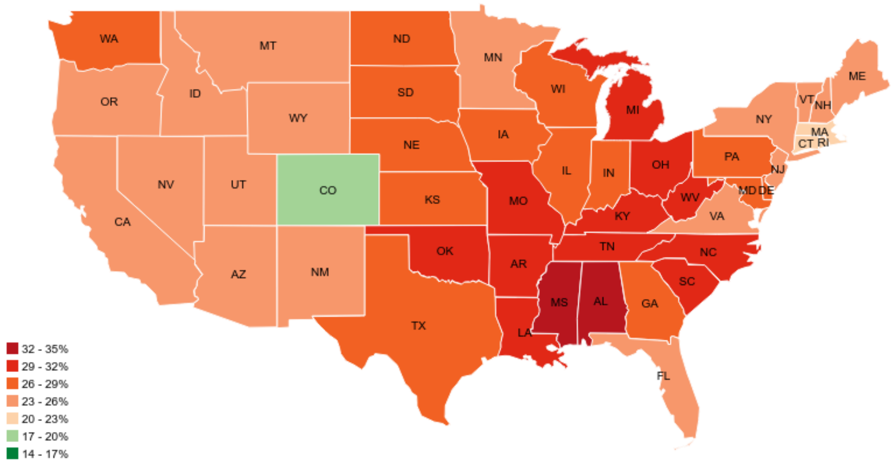


Fig. 2.2: Choropleth map of obese population, Body mass index (BMI) > 30, in the United States in 2008 (Chronic Disease Prevention and Promotion 2010).

2.4 Information Visualization of Hierarchic Data

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, as seen in Figure 2.3.

An common use case is a directory tree of a file system, e.g. a file browser or the command line utility `tree` on UNIX based operating systems. As Shneiderman (1992) mentions, this visualization becomes increasingly large when displaying more than one level and soon exceeds the entire screen size.

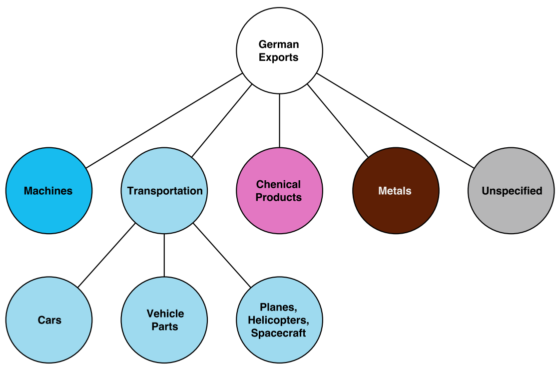


Fig. 2.3: Traditional visualization of a tree in form of a directed graph with edges and nodes and the root node at the top. This example visualizes equivalently a part of the treemap visualization of German exports in Figure 2.4.

2.4.1 Treemaps

Johnson and Shneiderman (1991) propose the treemap visualization technique, in which each node is a rectangle whose area is proportional to a specified dimension. In treemaps every node is visualized as a tile. The membership relationship is expressed with tiles containing other tiles, thus representing the hierarchy.

We can see two examples of treemaps in Figures 2.4 and 2.5. German exports are divided in generic groups like "Machines" and "Chemical Products" and include more specific groups like "Cars" and "Packaged Medicaments". The user can click on a drop-down menu to change the current level of hierarchy, only leaf nodes are displayed at a time.

The advantage of treemaps is that they are space-filling visualizations, i.e. they make 100% use of the available screen size. A treemap will, unlike a graph representation of a tree, never exceed the size of the screen.

The area of the tiles can be mapped to a data attribute, e.g. the file size on disk or, in cases of Figures 2.4 and 2.5, the percentage of export quota. Thus, a treemap can display even more information than a traditional graph representation. A disadvantage of treemaps is the variable size of each node. If more and more nodes are displayed, the size of each tile will get smaller and smaller and there might not be enough space to display a label. You can see this problem occur in Figure 2.5.

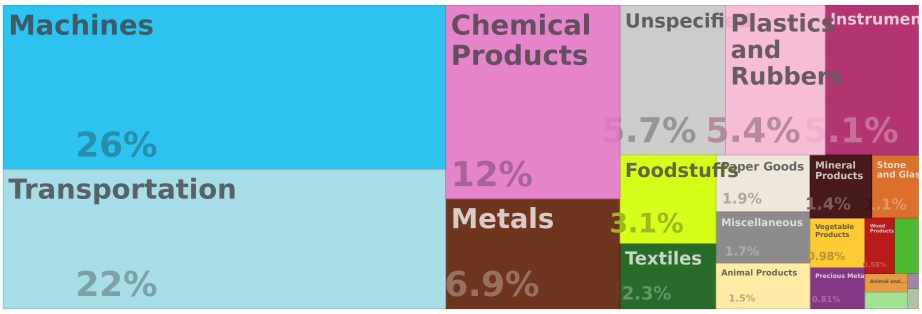


Fig. 2.4: A two dimensional treemap of Germany's foreign trade quota of exports, showing only the first hierarchy level (Observatory of Economic Complexity n.d.).

If treemaps are used to visualize geographic data like municipalities, real estates or streets, the placement of nodes depends on the tiling algorithm and not the geographic location on a map. Let's say a treemap visualizes a hierarchy of federal states and municipalities with the area of each tile mapped to the total population of administrative district. Even if the membership hierarchy is

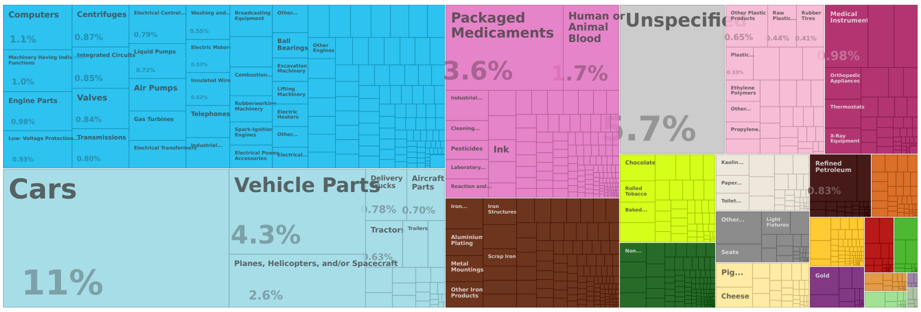


Fig. 2.5: Second level of detail of the treemap in Figure 2.4 (Observatory of Economic Complexity n.d.).

matched by the treemap, the placement of districts in the same hierarchy level is based on their total population and not their geographic location.

2.4.2 3D treemaps and 2.5D treemaps

3D treemaps are a concept introduced by Bladh, Carr, and Scholl (2004) in 2004. The authors transfer the concept of treemaps from two dimensional into three dimensional space, transforming tiles to blocks. They introduce "Step-Tree" (Bladh, Carr, and Scholl 2004), which is a three dimensional treemap to display a directory layout of a file system. It "differs from treemaps in that it employs three dimensions by stacking each subdirectory on top of its parent directory."

3D treemaps are superior to 2D treemaps for tasks with a pronounced topological challenge. User perform significantly better in interpreting the hierarchical structure. However, 3D visualizations also introduce some disadvantages. Blocks can superimpose each other, forcing the user to navigate the view point. The navigation of the view point itself is an increase of complexity not present in two dimensional treemaps.

The term 2.5D treemap was coined by Limberger et al. (2016) in 2016. A 2.5D treemap is just an ordinary 3D treemap, but it has all blocks attached to the ground, or more specifically, attached to the parent block. We can see an example of a 2.5D treemap in figure 2.6. All three dimensional treemaps in this thesis are in fact 2.5D treemaps and therefore this term is used for the rest of this work.

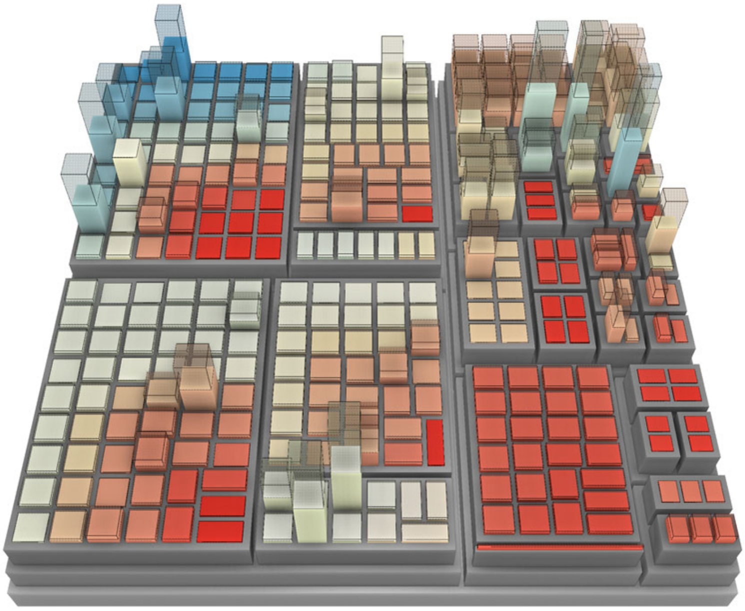


Fig. 2.6: Example of a 2.5D treemap (Döllner 2017).

2.5 Coordinated Multiple Views

Coordinated multiple views are a combination of data visualizations of the same data set in multiple views, often side-by-side. According to Roberts (2007) coordinated multiple views is just “a specific exploratory visualization technique that enables users to explore their data”. Because “the overall premise for the technique is that users understand their data better if they interact with the presented information and view it through different representations.” (Roberts 2007)

We can see an example of a coordinated multiple view in Figure 2.7. It displays spatial and temporal attributes of pictures from a picture database, as well as continuous attributes like popularity and number of comments. The user can move the mouse cursor over each item in the scatter plot and the graduated symbol map and the corresponding item is highlighted with a larger stroke in all other views. On the time line below, the user can also filter for pictures in a certain time frame by dragging the mouse from lower to upper limit.

2.5.1 Brushing and Linking

Brushing and linking is a common interaction pattern found in coordinated multiple views and often a crucial part of these visualizations. “The technique of

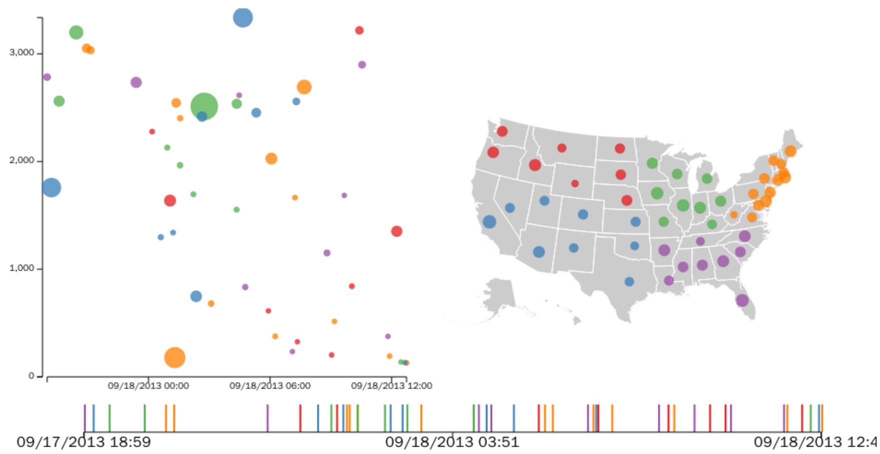


Fig. 2.7: Coordinated Multiple View (CMV) displaying the popularity, number of comments, location and time of pictures in a picture data base (Dukevis 2017).

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.” (Roberts 2007)

We can see an example in Figure 2.8. It displays an on-time performance of airlines, visualized with the “Crossfilter” javascript library.

This library is also one of the very few examples of an interaction framework for coordinated multiple views. However, this library is unmaintained as of December 2017, the most recent commit dating back to March 2016.

Each of the flight in the data set has an hour and a date for departure, an arrival delay, which can also be negative, and a traveled distance in miles. The user can “brush” the data by selecting an interval by dragging the mouse. The respective view will become a primary view and display the deselected items with a grey colour.

All other views become secondary views and display only selected items. The visualization takes the most recent 80 flights from the database that match all given filters. The user can further filter for items by dragging another interval in one of the secondary views.

This technique of propagating interactions to other views is called “linking”.

Figure 2.8 shows a filter for travels with a long delay, i.e. from 120 minutes to the maximum value, see the selection in the upper center. In the view in the upper left corner in Figure 2.8, we can see a correlation of long delays with the time of the day.

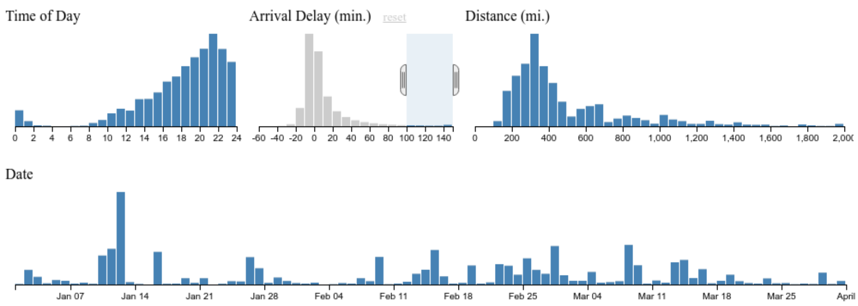


Fig. 2.8: Airline on-time performance: Correlation of time of day with arrival delay. Most recent flight with a delay of more than 100 minutes selected (Bostock 2017).

2.6 Technologies

This section is a stub, because I guess it will be removed. Do we actually need to introduce these technologies in this chapter?

Leaflet Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps (Agafonkin and contributors 2017). It has support for GeoJSON which makes it very easy to display tiled web maps with interactive overlays.

Web components is a recent standard of the W3C ((W3C) 2017) 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 (Li, Stockwell, and Mutton 2017). 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 (Wikipedia 2017). 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 (Katz 2017), 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 (Inc. 2017). 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.

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. 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.

PubSubJS is a topic-based publish/subscribe library written in JavaScript (Roderick 2017). 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.

2.7 Interaction Theory

According to Ho (2013) interactions are a crucial part of data visualizations, yet most research in the area 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?

The following sections will give an overview of the current research of interaction aspects for each group.

2.8 Interaction Categories

This section covers the larger part of research on interactions in coordinated multiple views: High-level classification and categorization.

In 1996 Shneiderman (1996) 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 (1998) 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 chart, (5) *change representation* of the *same data* by switching the chart type, (6) *link representations* to determine the relationship between items.

In 2002, Keim (2002) 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) drill-down operations to preserve an overview of the data are 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 (2007) 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* to show more or less detail, (6) *filter* to show something conditionally, (7) *connect* to show related items.

It is noticeable that all of these classifications of interactions are redundant. In this work the classification of Yi, Kang, and Stasko (2007) is used for the remaining parts because it is the most recent classification and it is based on the precursors.

Shall I give an example for each category of Yi, Kang, and Stasko (2007) classification?

2.9 Formalization of Interactions

This section covers the smaller part of research in coordinated multiple views, research that is *not* related to a high-level classification of interactions. Not only interactions in coordinated multiple views are considered but any kind of a formalization of interaction that may be used as the starting point for a framework for coordinated multiple views.

ITlib (Figueroa, Green, and Watson 2001) 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" (Figueroa, Green, and Watson 2001, 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, which is the common goal in data visualizations. In this case, the data model is the 3D scene and the goal is to manipulate the 3D scene.

Most importantly, the framework describes interaction techniques for a single viewpoint but not for coordinated multiple views.

A framework for Focus+Context Visualization by Bjork, Holmquist, and Redstrom (1999) is one of the few formalizations of interactions in data visualizations.

The idea behind the concept of focus and context visualization is to present the object of primary interest in full detail while at the same time giving a overview of the surroundings.

The authors of the paper distinguish first-level and second-level information visualizations:

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.

If V is affected by I the user can manipulate IV in order to change the way $[D]$ is represented. This statement holds e.g. for an interaction in which the user increases the visible level of hierarchy in a treemap, which is an `abstract/elaborate` interaction according to Yi, Kang, and Stasko (2007). The effect of such an interaction is depicted by the change from Figure 2.4 to Figure 2.5 in Section 2.5.

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. Regions of primary interest are distorted to appear magnified, while the remaining regions are minified.

The provided formalizations are a good starting point. Yet, they only consider interactions in the domain of focus and context visualizations, not in data visualization in general.

3

Analysis

The current treemap implementation and its set of possible interactions is described in Section 3.1. For the implementation of the geographic visualization several web frameworks are compared and advantages and disadvantages are contrasted in Section 3.3.

Section 3.4 describes essential characteristics of interactions in single and multiple data visualizations. These characteristics are relevant for a formalized language of coordinated multiple views. To accomplish this goal, the approach is to deduce the characteristics by a list of examples. What are the expected data structures for each visualization? Which visual variables can be used to show the effect of an interaction? A list of possible interactions is given for each visualization. Those interactions will be classified according to Yi, Kang, and Stasko (2007) and the relevant subject of the interaction is specified.

3.1 Existing Interactions

The current implementation comes with a visualization of a 2.5D treemap and should be complemented with a geographic visualization. In this section, the set of already possible interactions are classified according to Yi, Kang, and Stasko (2007) as in Section 2.8. They fall into these categories: *select*, *explore*, *reconfigure*, *encode* and *filter*.

The user can *select* one item in the view by clicking on it. The user can reveal a tooltip showing the item properties by moving the mouse cursor on the item, which is another *select* interaction.

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 the 2.5D treemap, like a turntable. The zoom factor can be changed by scrolling with the mouse on the canvas of the map.

Encode and *reconfigure* techniques are performed through a menu: The user can *reconfigure* different data sets and the displayed diagram, e.g. a treemap visualization based on the geometry shape, cubes or voronoi regions. In a sub-menu the user can *encode* properties of a data set in predefined visual variables, e.g. the height, color and texture of an item. A slider can be used to *filter* the data set on a data attribute. When the user drags the slider and changes an upper or a lower limit, items with an attribute beyond this interval are filtered out.

3.2 Software Requirements for CMVs

The existing implementation should be complemented with a coordinated multiple view layout. Next to the 2.5D treemap a geographic visualization should appear, so that the user can understand the geographic context of items in the treemap.

To ensure a future use of the software, this section provides a list of software requirements. Later in Chapter 6 these requirements will be reused to evaluate the system.

1. Serialization

   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.

2. Reversibility

   Reversibility means, in the context of coordinated multiple views, the ability to undo the effect of an interaction. Ideally, every interaction should be undoable. If not every interaction is undoable, the cost to replay the interactions from the original state up to the point of the interaction should be minimized.

3. Software Extensibility

   Software extensibility means, in this thesis, the costs of changing behaviour. How many views need to be touched, if a new interaction option is added to the system? How much time and effort is necessary to implement the new feature?

4. Maintainability

   Maintainability means, in this thesis, the costs of changing the source code. How error-prone is the system, will other views be affected if a new interaction is added to the system?

3.3 Component pattern and Web Frameworks

The current VISUAL ANALYTICS PLATFORM is a web application and the complementing CMV extension should be based on web technologies, too. Many popular web frameworks like Angular, Ember, React and Vue have developed mechanisms to update UI elements during user interactions. Because a view in a coordinated multiple view is a UI element as well, it is worth to consider one of these web frameworks for the implementation of the coordinated multiple view.

One software pattern, which is a commonly used update mechanism e.g. in Ember and React, is the so-called “component” pattern. It has become so widespread and prevalent that it triggered even a web specification called “web components”. This section evaluates the most suitable component based JavaScript framework for coordinated multiple views and whether or not to follow the “web components” web specification.

Three JavaScript frameworks have been evaluated: (1) GlimmerJS (2) Google Polymer and (3) ReactJS. GlimmerJS is the templating engine of Ember, the framework which the author of this thesis is most familiar with. Applications written with GlimmerJS can be built and exported as web components. Google Polymer is the most popular framework for web components. React is another very popular framework developed by Facebook, but does not support web components at all.

Most importantly, coordinated multiple views require a way 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 3.1.

To pass rich data to components however, web component frameworks have to roll their own data flow and syntax. Google Polymer’s own syntax to pass rich data to a components is shown in Listing 3.2. But this is a custom solution that abandons standard HTML.

Listing 3.1: An example of string based attributes of web components (Bidelman 2017).

```
<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 3.2: An small syntax example how Google Polymer passes rich data to a component.

```
<some-component some-prop="{{richData}}"></some-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. Custom solutions like the one of Polymer reduce the main motivation of implementing against web components: Platform agnostic flexibility.

As a summary, if there is (1) no obligation to implement web components and (2) an easy integration into an existing application is necessary, then React is the perfect choice for coordinated multiple views.

Table 3.1 shows the pros and cons of each framework for the use case.

Table 3.1: Comparison of component based web frameworks, advantages highlighted in green, disadvantages highlighted in red.

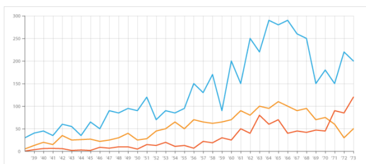
Web Components		Specific Framework
standard	string-based attributes	no standard
GlimmerJS	Google Polymer	React
familiarity	maturity	maturity
	documentation	documentation
	community	large community
		declarative style
		small size
		integrability

3.4 Arbitrary Data Visualization Techniques

Even though the use case of the coordinated multiple view is focused on a 2.5D treemap and a geographic visualization, the implementation should allow any kind of data visualizations to be added and coordinated as well. Instead of implementing as many data visualizations as possible, the approach in this thesis is to analyse many data visualization techniques theoretically. The goal is to come up with a set of formal requirements and a specification that anticipates additional data visualizations. With regard to the software requirements in Section 3.2, the coordinated multiple view framework needs to be extensible and maintainable.

The data visualization catalogue by Severino Ribecca lists many of the most used data visualizations (Ribecca 2017). This section covers a selection of data visualization techniques from that catalogue. For each technique the expected data structure is examined and a list of plausible interactions is systematically analysed. The gathered knowledge is a preparation of the conceptual framework in Section 4.

3.4.1 Line Graphs



(a) Line graphs

Fig. 3.1: Line graphs are used to display trends.

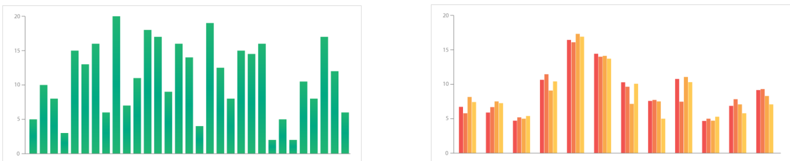
Line graphs display how quantitative values have changed over time. You can see an example in Figure 3.1. They are perfectly suited to show trends or compare multiple series of data with each other. Line graphs visualize one or many series of data in parallel and therefore the expected data format is *tabular*.

Line graphs are drawn in a Cartesian coordinate system, connecting subsequent points to each other. Thus, (1) position (2) orientation and (3) texture are constrained by the visualization technique. However, an interaction with the line graph can alter the (1) shape (2) color or (3) size of lines to communicate an interaction. It is further possible to highlight either the entire series of data or a single data point within that series, e.g. changing the shape and size of the point. Table 3.2 shows a list of plausible interactions in a line graph.

Table 3.2: Plausible interactions for line graphs.

Category	Description	Required information
Select	Highlight a data point	id of data point
Select	Highlight a data series	id of data series
Encode	Change colours of data series	id of data series + colour
Filter	Restrict interval on x-axis	lower limit + upper limit
Filter	Hide a data series	id of data series

3.4.2 Bar Charts and Multi-Set Bar Charts



(a) Bar Chart or column graph **(b)** Multi set bar Chart

Fig. 3.2: A multi set bar charts is a variation of a bar chart (Ribecca 2017).

Bar charts use either horizontal or vertical bars to show discrete, numerical comparisons across categories. The length of a bar displays a quantitative value of a category. You can see two examples in Figure 3.2.

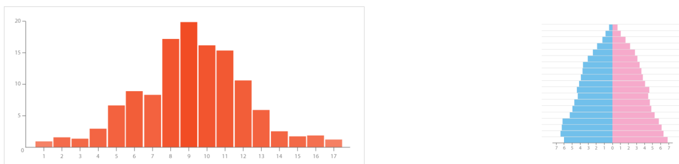
Multiple bar charts display many data series next to each other. Every series is grouped by category and a colour can be used to identify a data series. Like line graphs, bar charts expect a *tabular* data format. In contrast to line graphs, bar charts are used to show a comparison rather than a trend.

The type of the visualization constrains the (1) shape, (2) size and, in case of a multi set bar charts, (3) the colour of the visualization. An interaction can be shown by altering (1) position, (2) colour, (3) shape and (4) the texture of bars and columns. Table 3.3 lists some possible interactions.

Table 3.3: Plausible interactions for bar charts.

Category	Description	Required information
Select	Highlight a bar	id of data point
Encode	Change colours of data series	id of data serie(s) + colour(s)
Reconfigure	Sort by attribute	name of data attribute
Reconfigure	Drag bars to reorder data series	ordered list of ids of data series
Filter	Hide a data series	id of data series

3.4.3 Histograms



(a) Histogram **(b)** Population Pyramid

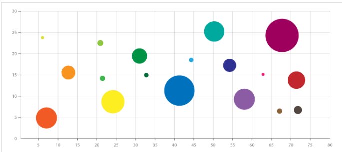
Fig. 3.3: A histogram is a bar chart over a continuous interval (Ribecca 2017).

Histograms, as shown in Figure 3.3, visualize the distribution of data over a continuous interval or a certain time period. A special type is the population pyramid, which is a pair of back-to-back histograms, one for each sex.

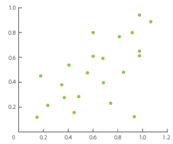
Histograms and bar charts expect the same kind of data, i.e. a *tabular* data structure. Almost the same interactions as in Table 3.3 can be applied to histograms, except a re-ordering of bars along the x-axis. This is not possible, because the histogram constrains the position of bars along the interval.

3.4.4 Bubble Charts and Scatter Plots

Both bubble charts and scatter plots are techniques to visualize continuous values from two data attributes. Points are placed with the two attributes in Cartesian coordinates in order to detect relationships and correlations. An example for each technique is displayed in Figure 3.4.



(a) Bubble Chart



(b) Scatter plot

Fig. 3.4: Bubble charts and scatter plots are similar regarding interactions (Ribecca 2017).

In case of bubble charts, each point is displayed as a bubble with a third value encoded in the size of bubbles. It is even possible to encode a fourth value in the colour of the bubble.

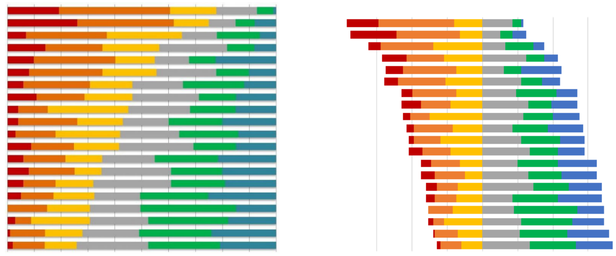
Like line graphs, bar charts and histograms, a scatter plot expects *tabular* data. Each data point can take up to four values (in case of a coloured bubble chart). As we can see in Table 3.4, interactions also include a zooming and movement of the viewpoint.

Table 3.4: Plausible interactions for bubble charts.

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

3.4.5 Stacked Bar Charts

Unlike a multi-set bar graph which displays bars side-by-side, stacked bar graphs segment their bars of multiple datasets on top of each other. A baseline, as



(a) Stacked bar chart **(b)** .. 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 (Mann 2016) (Peltier 2016).

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. A stacked bar chart also expects *tabular* data.

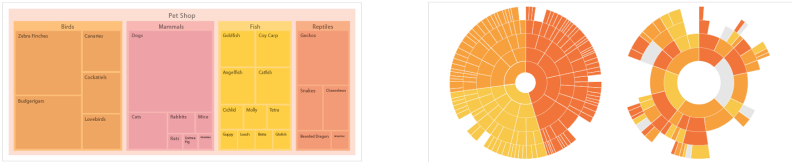
If the stacked bar chart has a baseline, often the algebraic sign of the numeric value defines the placement of the segment on the left or on the right side. Table 3.5 shows possible interactions, including the highlighting of a data point, a change of color mapping or a reordering of the baseline.

Table 3.5: Plausible interactions for stacked bar charts.

Category	Description	Required information
Select	Highlight a bar	id of data point
Encode	Change colour mapping	id of data series + colour
Reconfigure	Sort by attribute	name of data attribute
Reconfigure	Specify stacking order	ordered list of ids of data series
Reconfigure	Flip data series	list of ids of data series
Filter	Hide a data series	id of data series

3.4.6 Hierarchical visualizations

Tree maps are great to show hierarchical data without ever exceeding the available screen. Each data point is represented as a tile. Unlike a treemap a hierarchical ring diagram or sunburst diagram shows each level of the underlying tree as a series of rings.



(a) Tree map **(b)** Sunburst diagram

Fig. 3.6: Tree maps and sunburst diagrams are ideal to show hierarchies (Ribecca 2017).

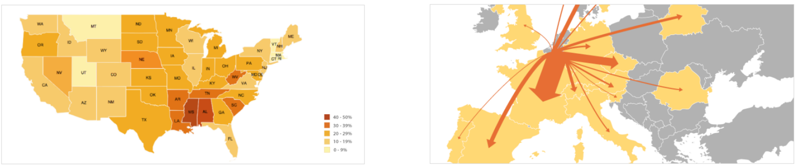
Therefore, both treemap and ring diagram expect *hierarchic* data. Typically, each node will have at least one continuous value that can be used as input for the tiling algorithm or layout algorithm respectively. Additionally, each node can encode more attributes by colour.

As these visualization techniques are about hierarchies, the visible, maximal depth of the tree may be increased or decreased. Again, interactions could include a highlighting of data points 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 child nodes of the parent node. Table 3.6 gives a more comprehensive list of interactions.

Table 3.6: Plausible interactions for hierarchical visualizations.

Category	**Description**	**Required information**
Select	Highlight a node	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	id of data series + colour
Reconfigure	Change data attribute used for layout	name of data attribute
Reconfigure	Sort by attribute	name of data attribute
Reconfigure	Specify order	ordered list of ids of data points
Abstract/Elaborate	Specify maximum depth of visible tree	number of hierarchy levels

3.4.7 Geographic Data Visualizations



(a) Choropleth map **(b)** Flow map

Fig. 3.7: Choropleth maps focus on a density while flow maps show a migration of data (Ribecca 2017).

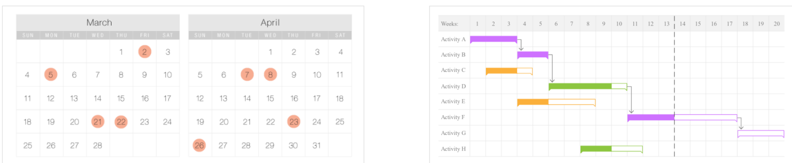
Choropleth maps and flow maps are thematic maps to visualize geographic data. Size, position and shape of a data point is determined by its geometry. Choropleth maps encode a continuous data attribute with relative values in the color of each region. Flow maps may display relationships between features, a data value defining the size, colour, direction or shape of each arrow. Figure 3.7 shows an example for each technique.

Non-geographic data may be given in a *tabular* form, assigned to each geographic feature. In contrast to tabular data, a flow map expects relationships between geographic features. Thus, it also expects *relational* data in form of a graph

Table 3.7: Plausible interactions for geographic visualizations.

Category	**Description**	**Required information**
Select	Highlight a feature	id of data point
Explore	Move viewport	latitude and longitude of viewpoint
Explore	Zoom in, zoom out	zoom factor
Encode	Change shape of marker	data id shape
Encode	Change mapping of category to colour	id of data series + colour
Encode	Change colour function	value + colour
Encode	Change data attribute used for colour	name of data attribute
Connect	Show relations of a feature	id of data point
Abstract/Elaborate	Change granularity of displayed regions	number of hierarchy levels

Temporal Data Visualizations



(a) Calendar **(b)** Gantt chart

Fig. 3.8: Similar to a calendar, a gantt chart shows activities and the progress along a time line (Ribecca 2017).

Activity diagrams, like calendars and gantt charts, and timelines are temporal visualizations. Gantt charts and timelines have in common that each feature is represented as a rectangle, with the duration of the feature mapped to size and position. A calendar, as shown in Figure 3.8, may show the event as a single marker for brevity. 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. Calendars and gantt charts expect *tabular* data, although data points might reoccur on a regular schedule. So some data points, i.e. events, might repeat infinitely. A very common interaction is the filtering of the data set by selecting an interval over a timeline visualization, e.g. by dragging the mouse from left to right. A more comprehensive list is shown in Table **??**.

Table 3.8: Interactions for temporal visualizations.

Category	Description	Required information
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	id of data series + colour
Encode	Change data attribute used for colour	data attribute
Filter	Remove a calendar or a category	id of data series
Filter	Filter data set by time interval	upper and lower limit

4

Conceptual Framework

Based on the preparations in Chapter 3.4 this chapter specifies a conceptual framework for coordinated multiple views. The approach involves the definition of the terminology, a formalization of an interaction and the basic components of the conceptual framework. These components are derived from the interactions and the data structures discovered in Chapter 3.

4.1 The Conceptual Framework from the User Perspective

Figure 4.1 shows a component diagram of the conceptual framework. The user interacts via input devices, e.g. mouse or keyboard, with a computer. In order to see the effect of the interaction the user oberserves an output device like a screen.

The computer or the browser provides an Application programming interface (API) for these devices and the implementation of each view has access to this API. On the other side, views can communicate with each other through a coordinator. This is necessary in order to coordinate interactions.

Each view can have multiple triggers and multiple effects. A trigger is the handling of an event, caused by a user interaction, e.g. a mouse click. A effect is the change of the visual representation of the view, in order to communicate the interaction.

Each view is self-responsible for the implementation of triggers and effects. This is inevitable, as sometimes a view can not react to an interaction at all. E.g. a re-ordering in a parallel plot will not affect a scatter plot, where the position of items is contrained by coordinates.

Interactions of the same category also need to be distinguishable. E.g. the user could select a group of items with a bounding box. Additionally, the user moves the mouse cursor on an item within that group in order to highlight the item.

Therefore the message exchanged between views not only includes the interaction category and the relevant item but also an interaction purpose.

Every view can subscribe to named interactions at the coordinator. The coordinator notifies all subscribed views when a named interaction happens. In order trigger an interaction, the visualization simply publishes the named interaction at the coordinator. This pattern is known as the *Publish-subscribe pattern* and widely used in message queues.

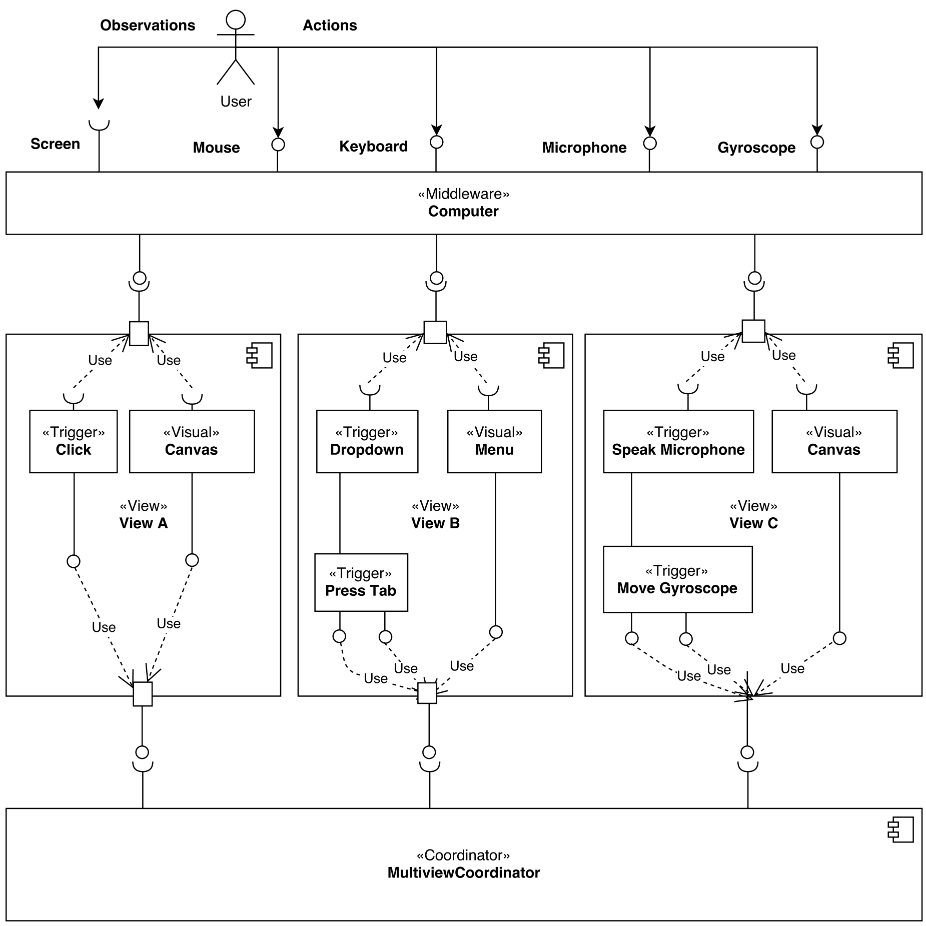


Fig. 4.1: Component diagram of the conceptual framework.

4.1.1 Interaction

In single views, an interaction I consists of at least a trigger and an effect:

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

The purpose of the interaction in a single view does not need to be explicitly stated as such. E.g. hovering over a geographic area changes the background colour of the polygon and the user can identify the interaction as a *highlighting*. This implicit meaning gets explicit in the described conceptual framework, so that views are able to react reasonably to interactions in other, outside views.

4.1.2 Interaction in Coordinated Multiple Views

An interaction I in coordinated multiple views is formalized as

$$I(T, M(C, P, S), E) \tag{4.2}$$

for a trigger T, a message M and the effect E. The message M is defined by a category C, an application specific purpose P as well as an interaction subject S. The shared part of the interaction among multiple views is the message M.

4.1.3 View

A view is the section of the screen in which the user can see a data visualization. A coordinated multiple view system shows multiple views, either side by side on a single screen or on different screens which are physically placed next to each other.

4.1.4 Trigger

If the event of a user action is handled by a view and causes an interaction, this handling is called a *trigger*. 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. As described in the introduction, views are responsible of their *triggers*.

4.1.5 Effect

The 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, e.g. a change of a visual variable according to Bertin (2010).

Some 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 treemap.

Similar to a *trigger*, a view is self-responsible of its visual *effects*. Obviously visual effects are not shared, as a visual variable might be constrained due to the nature of the visualization technique itself. A re-ordering interaction in a parallel plot will not have an effect in bubble charts, as position is constrained by the type of visualization.

4.1.6 Category

An category 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. Some examples of categories include: Selection, Deletion, Point-of-Interest, Filtering, Reordering, Re-encoding. Categories can be classified with the interaction categories of Yi, Kang, and Stasko (2007).

4.1.7 Purpose

The *purpose* describes the interaction in the context of a task or an application specific intention. 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 two interactions of the same category can be distinguished by a user-defined *purpose*.

4.1.8 Subject

A 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.

4.2 Shared data model

The shared data model is a model of the data on which all visualizations must agree on. To account for various data structures discovered in Chapter 3, we use an abstract data model that is inclusive enough to include tabular, hierarchical and relational data. You can see a class diagram of this data model in Figure 4.2.

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.

4.3 Encoding of Interaction Subjects

As described in Section 4.1.2, a message consists of the interaction *category*, *purpose* and *subject*. Category and purpose are just identifiers and can be encoded

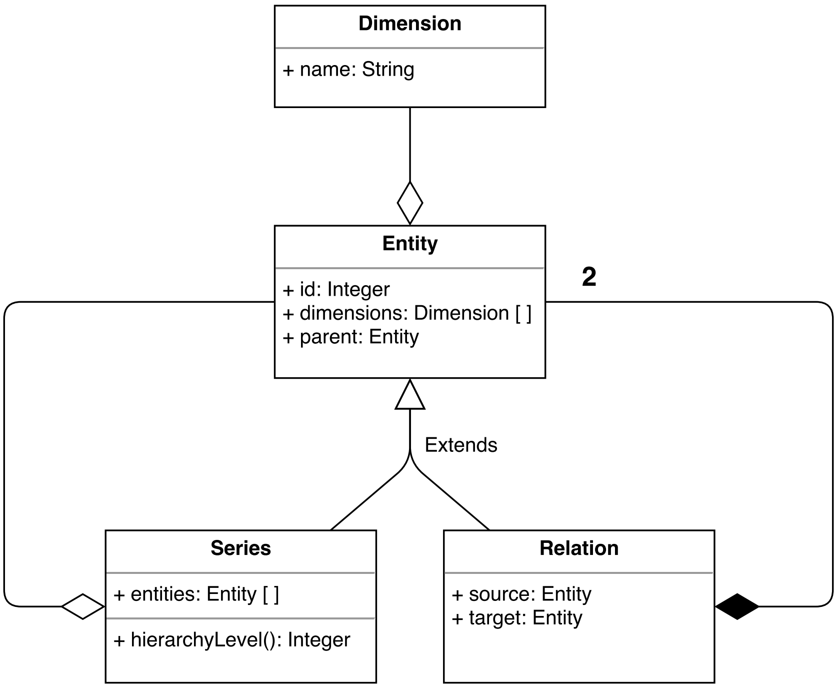


Fig. 4.2: A data structure for tabular, hierarchical and relational data.

with a number or a string. The subject is a highly variable object, that can be a number, a word, a list, a dictionary and so on. To account for diversity, the approach in this thesis is to model the subject as a mathematical function. This function is defined at runtime and the return type can be any of the classes already stated.

These functions operates on the ids of *entities*, *series*, and *relations* or their respective *attributes*. The name of the function is the *category* 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 4.2. To declare the functions explicitly, we define a couple of sets:

$$\mathbb{E} : \mathbb{E} \subseteq \mathbb{N} \tag{4.3}$$

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 $\mathbb{N}$.

$$\mathbb{D} : \mathbb{D} \subseteq \Sigma^* \tag{4.4}$$

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.

$$Space(\mathbb{D}) \tag{4.5}$$

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 $Space(D)$ also includes the set of all possible discrete values of dimension.

$$\mathbb{V} = \{x, y, y2, y3 \ldots yn, z, height, colour, size, shape, orientation \ldots\} \tag{4.6}$$

A discrete set to capture visual variables position, size, orientation, texture, colour and shape, as described in Section 2.3.1. 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 categories based on the aforementioned sets:

$$Select : \varnothing \to \mathcal{P}(\mathbb{E}) \tag{4.7}$$

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. treemaps or sunburst maps, may choose a single focused entity to be the root node of the currently displayed subtree.

$$Filter : \mathbb{E} \rightarrow \{\bot, \top\} \tag{4.8}$$

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.

$$Window : \mathbb{D} \rightarrow \mathcal{P}(Space(\mathbb{D})) \tag{4.9}$$

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 treemap.

$$Order : \mathbb{E} \times \mathbb{E} \rightarrow \mathbb{R} \tag{4.10}$$

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.

$$Encode : \mathbb{D} \to \mathbb{V} \tag{4.11}$$

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 treemaps. 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 message composed of a *Select* function and the category *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 category *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 category *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.

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 Implemented interactions

In the course of this thesis the following interactions have been implemented:

- Select
  - Highlighting an entity is possible by moving the mouse cursor on a geographic feature in the geographic visualization or a block in the 2.5D treemap. The corresponding block or geographic feature will be highlighted respectively.
  - A click on a block in the 2.5D treemap selects this entity in the geographic visualization and vice versa.
  - Holding the control key, multiple clicks on a block in the 2.5D treemap creates a group of selected entities. Corresponding geographic feature or blocks in the 2.5D treemap or geographic visualization are selected as well.
- Explore
  - Selecting a block in the 2.5D treemap centers the viewpoint in the geographic visualization on the corresponding geographic feature.

- Selecting a group of blocks in the 2.5D treemap centers the viewpoint in the geographic visualization on the boundaries of all corresponding geographic features.

- Reconfigure
  - Choosing a different data set updates the geometries in the geographic visualization.
  - Selecting another shape for the 2.5D treemap (e.g. point instead of polygon geometries) changes the visual representation in the geographic visualization.

If we still have time, add filter interaction here

5.2 Coordinated Multiple View Layout

Figure 5.1 shows the layout of the different views from the user perspective. The 2.5D treemap is on the left, the geographic visualization is on the right and some additional views to inspect and manually publish messages are displayed.

The currently visualized data set is called “Wahlkreise” and consists of German administrative districts. The 2.5D treemap is tiled based on unemployment rate and the colour is based on the percentage of high-school graduates. District “Magdeburg” is selected in the geographic visualization with an unusually high percentage of graduates compared to other districts with a similar unemployment rate.

5.3 Architecture

The architecture of the implementation is depicted in the class diagram in Figure 5.2. For each of the views in Figure 5.1 you can see a corresponding class in Figure 5.2, i.e. `Treemap`, `MapComponent`, `MessageLog` and `DebugView`.

All views have a reference to `MultiviewCoordinator` in order to subscribe to interactions. The rendering 2.5D treemap is controlled by `UAController`, thus it is connected to `MultiviewCoordinator` through this class. The `MultiviewCoordinator` itself does not have a visual representation.

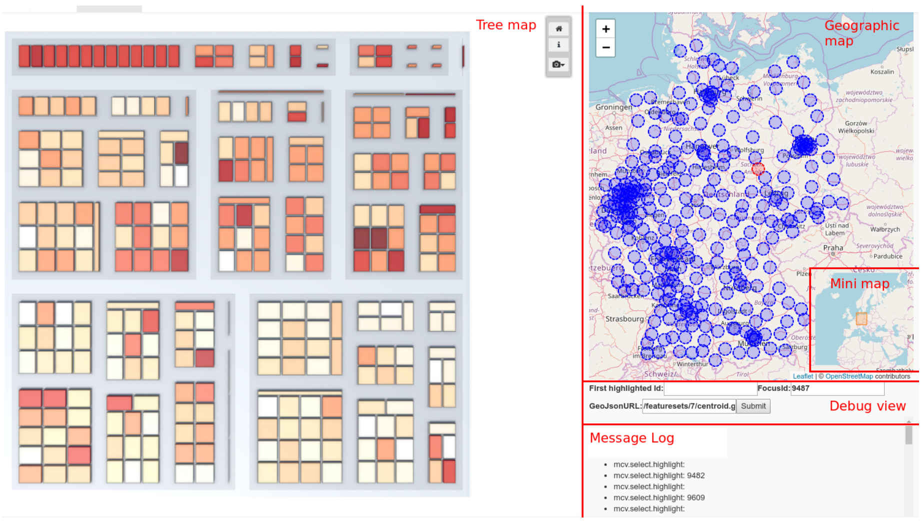


Fig. 5.1: This Figure shows the layout of the coordinated multiple view, red lines indicate the borders of different views.

5.3.1 Update mechanisms

In Figure 5.2 views with a `render` are implemented as React components and can be rendered in place of a node inside the Document Object Model (DOM) tree. React components and their dependant components get automatically re-rendered if their internal state changes.

E.g. if the position of the view point of the `MapComponent` changes, that will re-render the included `Map` which depends on the position. If the Geometries of the visualized data set are updated, that will change the `GeoJSON` component but not the `Map`.

5.4 Notifications

The automatic update mechanism works only within the virtual DOM of React components. Updates on view level need to be implemented manually.

In accordance to Chapter 4, the class `MultiviewCoordinator` implements the `publish-subscribe` pattern for coordination and works as a broker for interactions across coordinated multiple views. To achieve this, it uses the JavaScript library `PubSubJS` which is an topic-based, asynchronous implementation of the pattern.

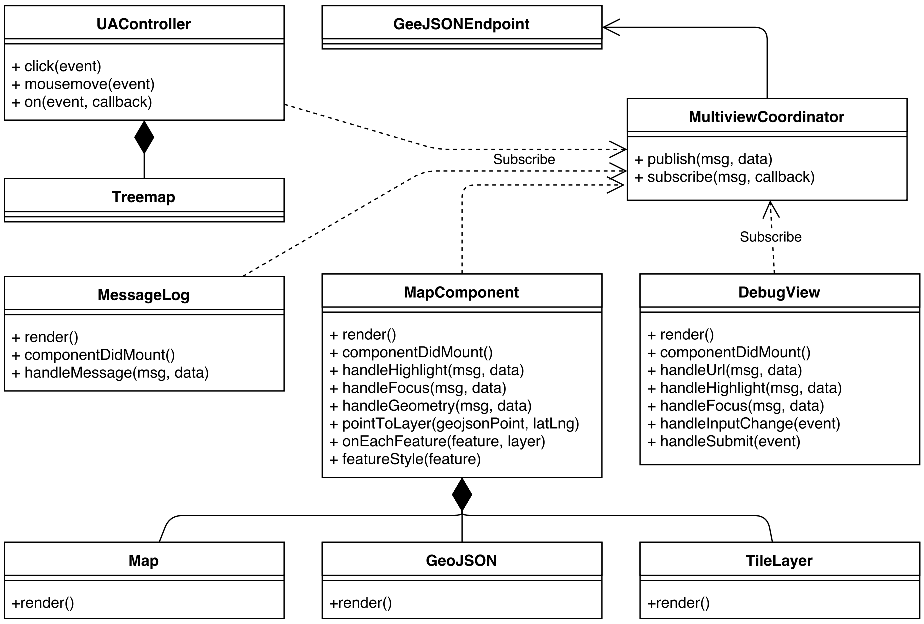


Fig. 5.2: Architecture of components. Every class with a `render` method is a React component. `MultiviewCoordinator` is used to coordinate 2.5D treemap and geographic visualization as well as to load geometry data.

Let's take an example from a common usage scenario: During initialization, all views subscribe to `MultiviewCoordinator` in order to re-render after interactions. Then the user chooses another data set and the geometries are updated. After that, the user clicks with the mouse on a geographic feature and the corresponding item is focused. You can see this scenario in the sequence diagram in Figure 5.3.

5.4.1 Encoding of interaction category and purpose

Each view gets a reference to the coordinator, e.g. during initialization, and can subscribe to interactions as seen in Listing 5.1.

`PubSubJS` allows nested topics, so categories and purposes are encoded as nested strings. A view can subscribe to a group of interactions, e.g. the group `mcv.select` includes both `mcv.select.highlight` and `mcv.select.focus`. This approach gives each view self-responsibility of the implementation of individual interactions.

According to Chapter 4, categories and purposes are encoded with nested strings. Figure 5.4 gives an example of a select interaction with the purpose to focus on three entities.

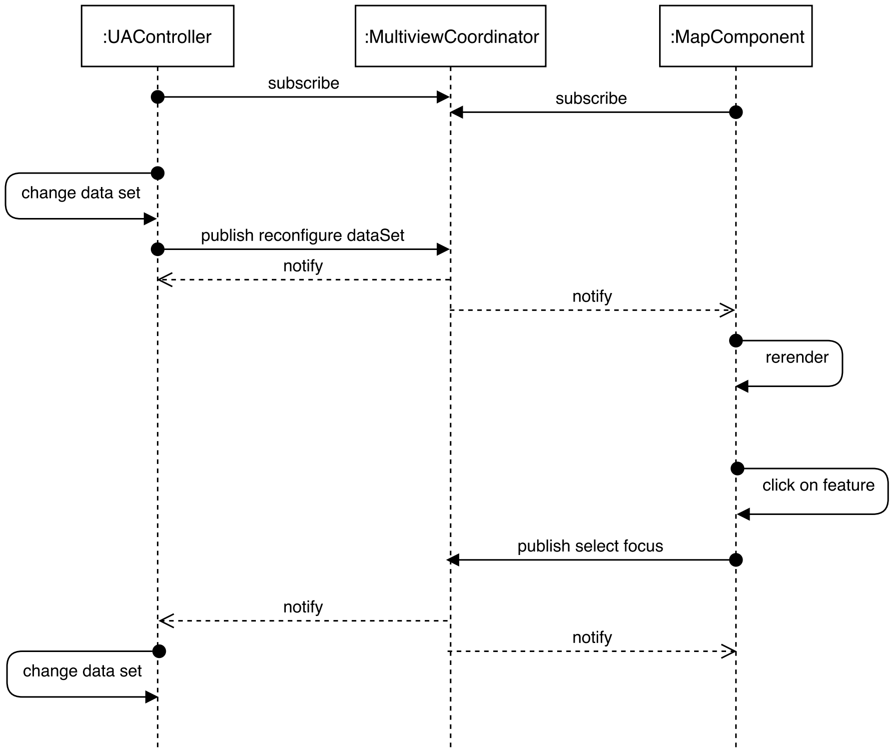


Fig. 5.3: This sequence diagram shows the notification of different components during a common usage scenario.

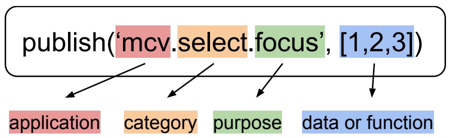


Fig. 5.4: Example of the encoding of interaction category and purpose.

Not only basic data types like numbers and strings can be published, as seen in Listing 5.1, but also functions. If an interaction of type `filter` is published, it is not necessary to publish all filtered ids of entities explicitly. Instead, the `data` of the interaction can be a threshold function which, applied on an entity, returns whether or not the entity is filtered.

Listing 5.1: A simplified example how to subscribe to an interaction.

```
let coordinator = new MultiviewCoordinator();

// subscribe to a `select` interaction with purpose `highlight`
coordinator.subscribe('mcv.select.highlight', (msg, data) => {
   console.log(msg, data);
});
```

```

// publish a topic asyncronously
coordinator.publish( 'mcv.select.highlight', 4711);
```

5.5 MultiviewCoordinator

As a central part of the implementation, the `MultiviewCoordinator` is also used for certain performance optimizations and data integrity considerations. Therefore, it is responsible to query geometry data.

Geometries are exchanged with `GeoJSON` as data format. GeoJSON is a format for encoding a variety of geographic data structures ((IETF) 2016). Based on JSON, it can represent simple geographic features like points, lines and areas and reserves a properties object for non-spatial attributes.

If the user selects a different data set, the `url` of the GeoJSON endpoint changes and an update of the geometry data is required. This behaviour is implemented by the `MultiviewCoordinator` which observes all changes to the `url` of the GeoJSON endpoint. If the `url` changes, the coordinator fetches geometry data, publishes a change of the geometries and all subscribed views can re-render. This is also a performance optimization, as it reduces the number of requests to the `GeoJSON` endpoint.

An example of these geometries can be seen in listing 5.2. 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 treemap 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.

Listing 5.2: A GeoJSON example of a user distribution across German federal states. Coordinates are omitted.

```
{
  "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
    }
  ]
}
```

5.6 Render views in the DOM

Adding the geographic visualization to the DOM is straightforward. Listing 5.3 shows an example how to use React's low-level API to render components inside the DOM. It uses a HTML id `multiview-map-component` to reference a particular node in the HTML document. The HTML document needs to import three JavaScript files, i.e. two imported libraries required by React and the compiled JavaScript application.

Listing 5.3: Example application written in TypeScript. Views can be added to the DOM individually. The implementation exposes convenient TypeScript declarations, here `MultiviewCoordinator` and `MapComponent` are imported.

```
1  import * as React from "react";
2  import * as ReactDOM from "react-dom";
3  import { MapComponent, MultiviewCoordinator} from 'urban-
       analytics-multiview-map-component';
4
5  let coordinator = new MultiviewCoordinator();
6
7  ReactDOM.render(
8    <MapComponent coordinator={coordinator}/>,
9    document.getElementById('multiview-map-component')
10 );
```

The implementation of the conceptual framework is written in `TypeScript`. TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It provides optional static typing, classes and interfaces and helps to reduce errors by raising type errors at compile time. The implementation exports typescript declaration and you can see an import of the classes `MapComponent` and `MultiviewCoordinator` on lines 1 to 3 in Listing 5.3.

5.7 Map Component

The geographical visualization makes use of three components of the React Leaflet library: `Map`, `TileLayer` and `GeoJSON`. Listing 5.4 shows the `render` method of the component.

Listing 5.4: `render` method of the Map component of the geographical visualization.

```
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>
  );
}
```

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 into 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." (Cam 2017).

The subscriptions to the coordinator happen during the method `didComponentMount`, a lifecycle callback method provided by React. You can see the respective interaction handlers `handleHighlight`, `handleFocus` and `handleGeometry` in the class diagram in Figure 5.2.

5.7.1 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 5.5 shows the event handlers added to the map.

Listing 5.5: `onEachFeature` callback, adding handlers for mouse events.

```
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)]);
      }
    }
  });
}
```

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 distinguished 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.

Listing 5.6: `featureStyle` callback, configuring the visual appearance depending on the currently highlighted or focused feature ids.

```
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 };
}
```

The `featureStyle` in Listing 5.6 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.

Listing 5.7: `pointToLayer` callback, if a feature of `GeoJSON` has a point geometry, it will be shown as a circle.

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

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.

6

Evaluation and Discussion

In this chapter the reference implementation of the conceptual framework is evaluated. First, a couple of use cases are described and how a geographical visualization can create more value. A performance evaluation is carried out, to discover technical limitations and to find performance bottlenecks. As a last step, the requirements of Chapter 3 are used to validate the conceptual framework and the reference implementation.

6.1 Use Cases and Discussion

The following three use case scenarios demonstrate how more insights can be gained with geographic visualization next to the 2.5D treemap.

6.1.1 Explain outliers

2.5D treemaps can have outliers, i.e. unusual local maxima of a certain attribute. If the attribute is mapped to the height of the corresponding block, a local maximum is identifiable by a block that protrudes from a group of evenly leveled blocks. We can see such an example in Figure 6.1.

The data set of the visualizations in the figure includes gas stations in Berlin. The 2.5D treemap is configured as follows: The layout is based on the brand name of a gas station, i.e. gas stations of the same brand like “Total” or “Aral” are grouped together. Height and colour of the blocks are mapped to the price of “Diesel” and “E10” respectively.

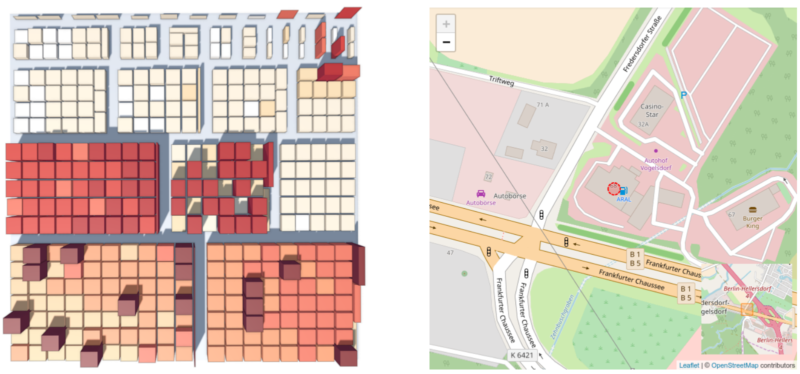
A 2.5D treemap with this configuration is suited to show a correlation of brand and price. Gas stations of brand “Total” are located on the center-left side of the 2.5D treemap and they are rather expensive in general but with smaller price variations.

This is in strong contrast to gas stations of brand “ARAL” in the lower left: 10 gas stations can be identified as outliers, which are more expensive than other gas stations in the group. “Esso” and “Shell”, located in the center and on the lower right, show outliers, too.

Gas stations of the brands “HEM”, “Star” and “Sprint” are generally inexpensive and located in the groups with lighter colours at the top and on the right.

However, the 2.5D treemap alone is not able to explain the reason for certain outliers. What is special about the protruding blocks, i.e. the outliers within a group?

The geographic visualization can give a possible explanation: Many of those outliers are gas stations located next to a highway. These gas stations are slightly more expensive in general and significantly more expensive if they belong to the brand “ARAL”.



(a) Tree map **(b)** Point of interest visualization

Fig. 6.1: Expensive gas stations, compared to other gas stations of the same brand, are often located next to highways.

6.1.2 Multi-Select in 2.5D treemap

This use case demonstrates the benefit of a multiple select and the ability to relate interesting features with their geographic context. Figure 6.2 (a) shows a 2.5D treemap with the following configuration: The layout is based on the population density, large clusters on the right are sparsely populated districts.

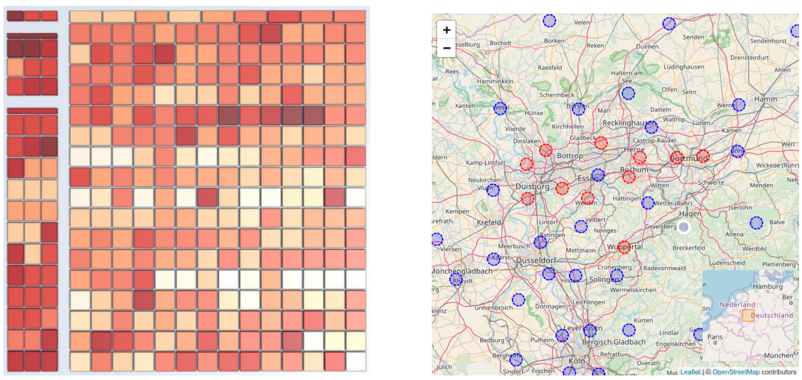
Colour is based on the increase or decrease of inhabitants, a red colour indicating a growth of inhabitants.

There are three groups of items in the 2.5D treemap that catch our interest: (1) A group of orange items on the left side of treemap, (2) many scattered, white coloured items in the large cluster on the right and (3) three purple coloured items in the large cluster on the right.

Districts in the orange group are rather densely populated and show a decline of population. But as we can see in Figure 6.2 (b) the districts of this group have a geographic context: They are all located in the Ruhr area.

A similar observation can be made about the white group, i.e. sparsely populated districts with a serious decline of population: These districts are all located in the east of Germany, see Figure 6.3 (a).

And finally, even the three purple districts in the cluster of sparsely populated districts are geographically related as well: These districts are in the vicinity of Munich, as we can see in Figure 6.3 (b).



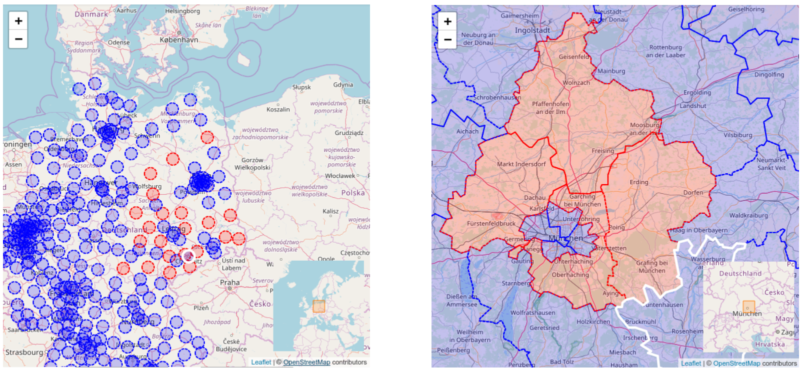
(a) Tree map **(b)** Orange group

Fig. 6.2: The orange coloured group on the left, districts with a moderate decline of inhabitants and large population density, are all districts of the Ruhr area.

6.1.3 Bounding Box Selection in geographic visualization

Similar to the multiple select in Section 6.1.2, multiple selects can be carried out by a bounding box in the geographic visualization.

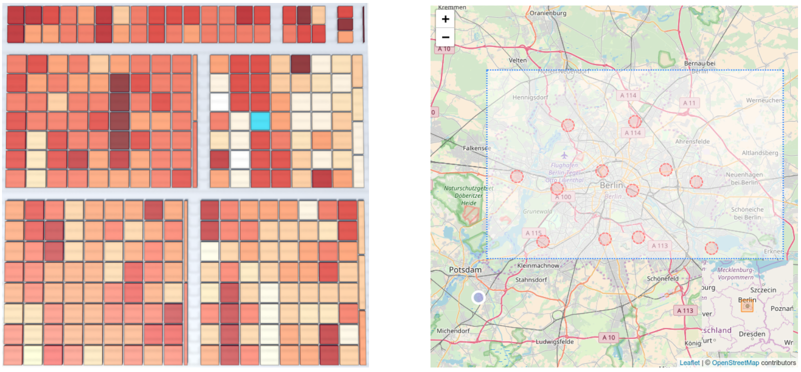
The layout of the 2.5D treemap in Figure 6.4 (a) is based on construction activity, i.e. the number of completed accommodations per capita. The colour is



(a) White group **(b)** Purple group

Fig. 6.3: The sparsely populated districts with a serious decline of inhabitants are all located in the east of Germany. The few districts with a low population density and high increase of population are all located in the vicinity of Munich.

mapped to the increase of population. As Berlin is known for quickly rising rents and real estate speculation we select all districts in Berlin with a bounding box. All selected districts are placed in the center right group of the 2.5D treemap. They are the dark red items next to the currently highlighted item in Figure 6.4 (a). But as we can see in Figure 6.4 (a), the districts are not in the group with the highest construction activity at the very top.



(a) Tree map **(b)** Geographic map

Fig. 6.4: A bounding box multiple select in the geographic visualization can reveal a non-geographic context in the 2.5D treemap

6.1.4 Use case summary

The scenario in Section 6.1.1 is a great example how a geographic visualization can create more insights. The 2.5D treemap alone allows to identify the outliers and also shows a possible correlation of price and brand. However, without the geographic visualization it is not possible to draw a hypothesis of the proximity to a highway being the reason for the high prices. The geographic context is just not present in the raw data.

Deliberately selecting a particular area of interest in the geographic visualization by clicking on an interesting item in the 2.5D treemap can be considered to be a new technique.

Similarly, in the scenario in Section 6.1.2, the 2.5D treemap alone can show outliers or group of outliers. However, it is not possible to see the geographic relation between those items in that group immediately. The geographic visualization makes the geographic context visible. In this scenario, the multiple selection really helps to select items of the white group, which is scattered across the 2.5D treemap.

Finally, in the scenario of Section 6.1.3, we can also do the reverse: Identify a non-geographic relation in the 2.5D treemap by selecting geographically related items with a bounding box. In this case we can see a very similar construction activity.

6.2 Performance Evaluation and Limitations

The following performance evaluation was carried out with the built-in runtime performance analysis feature of the Chrome browser. In particular, a Chromium Browser was used in Version 62.0 (64 Bit). The hardware specifications of the machine are listed in Table 6.1.

Table 6.1: Hardware specifications.

Device name:	LENOVO ThinkPad L540
CPU type:	Intel i3-4100M CPU @ 2.50GHz
#CPUs:	4
Main memory:	8GiB
Graphics card:	Intel 4th Gen Core Processor Integrated Graphics Controller

A couple of data sets were used in three different scenarios: (1) The 2.5D treemap without a geographical visualization, just publishing interactions, (2) an example application of the geographical visualization without a 2.5D treemap and (3) both visualizations together.

In the first and second scenario, the data set is loaded, some data points are highlighted and then some data points are focused with a single-select and a multi-select holding the control-key. In the second scenario and third scenarios, which have a geographical visualization, we also select many items with a select box, holding the shift-key. For every scenario there are six data sets, that is 18 profilings, and each scenario profiling took about 60 seconds to finish.

Table 6.2 shows the list of data sets used for profiling the performance of the reference implementation. The largest data set consists of German administrative districts called “Landkreise Deutschland” with a total size of 2.13 MiB. The data set with the highest number of features is called “Immoscout Wohnungsangebote” with 8601 coordinates German real estates, totalling 2.11 MiB.

Table 6.2: Data sets used for performance profiling.

Data Set	Features	Type	Size (MiB)
Bundesländer Deutschland	16	Areas	0.64
Tankstellen Berlin	366	Points	0.75
Wahlkreise BT 2009	299	Areas	0.91
Regierungsbezirke Deutschland	31	Areas	0.94
Immoscout Wohnungsangebote	8601	Points	2.11
Landkreise Deutschland	402	Areas	2.13

6.2.1 Immoscout

The slowest profiling is the visualization of data set “Immoscout”. Chrome’s runtime analysis shows a red bar at the top of the screen if the frames per second drop in such a way that it impairs the perceived interactivity. You can see a screenshot of the analysis in Figure 6.5.

As you can see in Figure 6.6 the 2.5D treemap spends almost the entire CPU time in scripting. The geographic visualization has a more balanced CPU time, spending time for painting and rendering during focusing interactions.

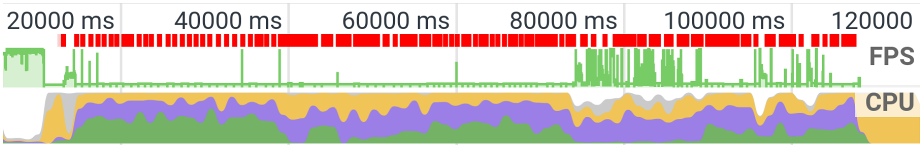


Fig. 6.5: During profiling of both 2.5D treemap and geographic visualization visualizing the "Immoscout" data set, the frame per second rate drops to 1 FPS.

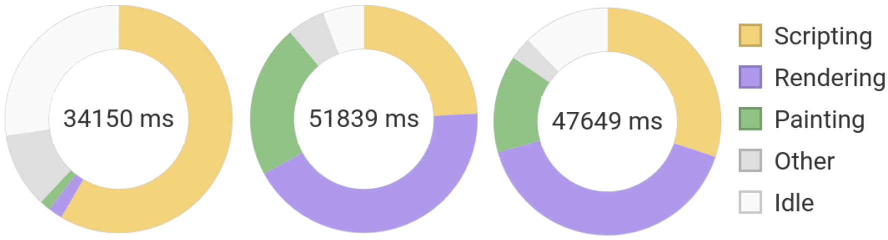


Fig. 6.6: Summary of profile data for data set "Immoscout" of 2.5D treemap only (left), both 2.5D treemap + geographic visualization (center) and only geographic visualization (right).

Going through the timeline, we can identify the handling of the "mousemove" event to be the likely cause of this slow scripting. The 2.5D treemap constantly checks the point or polygon which is under the mouse cursor. This is likely a performance problem.

The profile summary looks totally different for the scenario of a 2.5D treemap along with a geographical visualization. Compared to just a 2.5D treemap only, much more time is spent during painting and rendering. This is caused by the fact, that LeafletJS moves the viewpoint and zooms if a feature is focused. This can cause a network request and will re-render background tiles.

Note that not the communication between views hits the performance but rather the change of visual representation of views.

Figure 6.5 shows spikes whenever an interaction is made. Most of these interactions are `highlight` interactions, when the user moves the mouse cursor. Figure 6.7 shows the lower part of the callstack during such a `highlight` interaction.

We can identify the dominating subroutine to be `setStyle` which spans almost the entire callstack. Below this subroutine, you can see a lot of quick calls for each layer. `LeafletJS` iterates through all geographic features in order to update the style, e.g. change the stroke width. Therefore, `setStyle` seems to be costly operation for a large number of features. This would explain why many small features have a stronger performance impact than fewer but larger features.

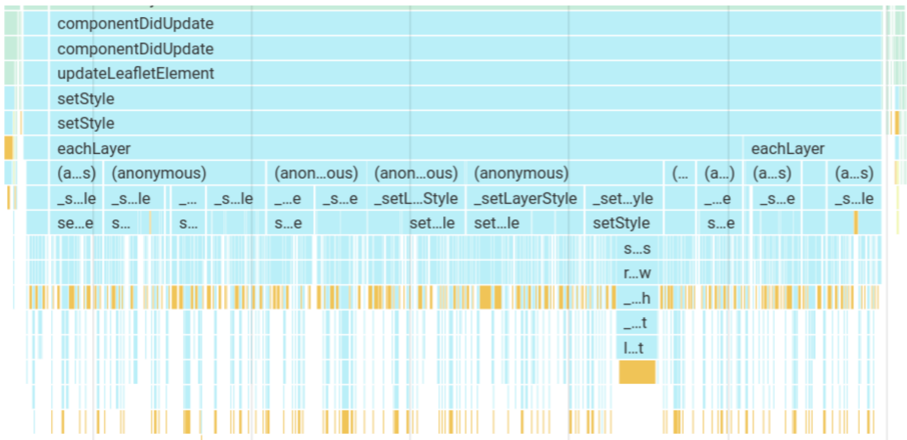
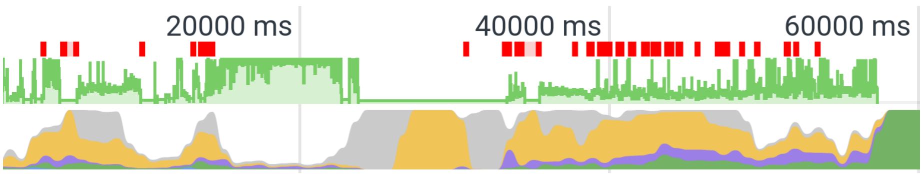


Fig. 6.7: The sawtooth pattern of the callstack during a `highlight` interaction indicates an expensive iteration of all features of the `GeoJSON`.

6.2.2 Landkreise

The profiling of data set “Landkreise” seems to support that assumption. This data set is larger than “Immoscout” but has fewer features. Nevertheless, the frame per second rate rarely drops in a way which has an impact on interactivity, as you can see in Figure 6.8.

Fig. 6.8: Larger but fewer features seem to have positive effects on the frame rate.



The geographic visualization alone idles almost 50% of the CPU time, as you can see on the right side of Figure 6.9.

6.3 Evaluation of Requirements

In this section, the reference implementation is evaluated based on the requirements from Section 3.2.

Serialization of interactions depends on the published interaction data. The interaction type is just a string and therefore trivial to serialize. But the payload

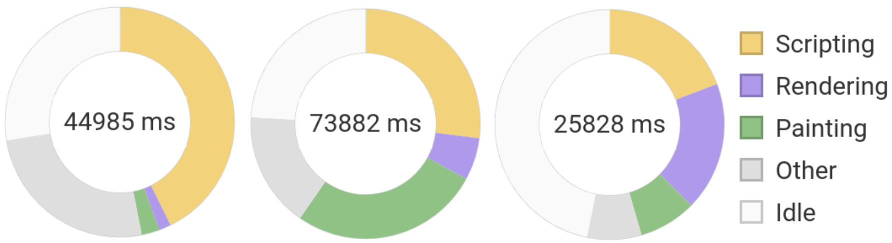


Fig. 6.9: Summary of profile data for data set “Landkreise” of 2.5D treemap only (left), both 2.5D treemap + geographic visualization (center) and only geographic visualization (right).

of the interaction can be an arbitrary JavaScript object, so the serialization depends on the serialization of that data.

In our case, the largest JavaScript object that is published as interaction data are the geometries during initialization. This step is necessary as there is no shared state of data between visualizations and every view needs to hold its own data. Only identifiers are valid across multiple views, but each view is allowed to have its own internal structure of data.

The largest data set of geometries at hand is called “Immoscout Wohnungsangebote” and consists of 8617 geographical features with a total size of 2.1 MB on disk. These geometries are stored as GeoJSON, wich is based on JavaScript Object Notation (JSON).

After the initialization, data transmission between views can be reduced by exchanging implicit functions instead of explicit data sets. E.g. if some data points shall be hidden, the threshold function can be published instead of a set of entity ids that are hidden. In JavaScript, functions can be serialized with `toString()` and deserialized with `eval`.

Reversibility of interactions also does not depend on the interaction framework but rather on the implementation of the participating visualizations. The undoing of an interaction needs to be implemented in each view separately. If the interaction framework is used to publish an `undo` interaction, there are two possible implementations: (1) Each visualizations keeps a record of every past interaction and can replay the record up to the desired step (2) or the mathematical inverse of the function is published. Both options have an undesired impact on the required memory, because data needs to be kept in memory for every visualization. If the record of interactions is long, the first implementation will take a long time to replay every interaction up to the desired state.

If past interactions are mathematically reversible, e.g. a threshold function for a filter interaction, the `undo` interaction only needs to publish the inverse function. The number of interactions does not have an impact on the performance, but if more than just one interaction shall be reversed, a record of past interactions need to be stored.

Data extensibility of interactions is enabled with arbitrary JavaScript objects. In Section 3.2 good extensibility is present when (1) additional data attributes can be added without lookup of corresponding items and (2) no de-duplication steps are necessary when new items are added. Since every entity in the data set is considered to be identifiable, a lookup of additional data can be accomplished a request with the entity id. Entity ids are unique, so no de-duplication is necessary. Since attributes have a unique name, the value of an attribute for the entity can be replaced without de-duplication.

Development costs do not diminish significantly. The framework design assumes loose coupling, independent views and no shared state. It is lightweight, has a low complexity and scales well. The downside of this approach is that it does not come with a huge simplification of implementing visual changes and event handlers in single views. Implementing the correct trigger and effect of interactions stays to be the main burden of work.

Maintainability as described in Section 3.2 means how much other parts of the code are impacted by an interaction and how error-prone the framework is. The framework benefits of the main advantages of the publish-subscribe pattern, i.e. loose coupling and scalability. On the other hand, it suffers from the main disadvantages of this software pattern, the decoupling of publisher and subscriber. Debugging the connections can be quite cumbersome and the asynchronous publishing can lead to further errors.

The geographical visualization is 873 lines long which is quite small. The glue code of a very simple example takes 132 lines of code.

7

Summary and Conclusion

This thesis demonstrates how coordinated multiple views can be used to improve comprehensibility and interactivity of 2.5D treemaps. In particular, the approach was applied on multi-dimensional, geographical data with one additional, coordinated geographical visualization. Apart from the specific use case, interactions in coordinated multiple views were formally described and a conceptual framework was developed that can be used for arbitrary data visualizations.

Data analysts can use the new knowledge by applying 2.5D treemaps in the context of decision support systems. This applies especially in application scenarios with a strong geographical context, e.g. local administration and urban planning. It was shown that a coordinated geographical visualization can improve usability by providing orientation and a better recognition of regions and locations in 2.5D treemaps. The enrichment of the original data set by adding geographical information, which is not present in the data set itself, turned out to be a great improvement.

The formalization of an interaction in coordinated multiple views and the subsequent development of a conceptual framework is helpful for researchers and developers of visualization frameworks. While coordinated multiple views are a well researched topic, research was missing regarding a formalization of coordinated interactions. Developers can built frameworks based on the conceptual framework specified in Chapter 4.

As described in Chapter 6, the framework is lightweight, has good scalability and loose coupling. With respect to the requirements of a framework of coordinated multiple views it shows good serialization and data extensibility. But the framework is responsible only to coordinate interactions, so the main effort of implementing interactions is still there.

7.1 Future Work

In the future, the conceptual framework should be validated and tested with more data visualizations. Chapter 3 has a long list of examples of interactions which can be implemented for validation.

Chapter 6 suggests two approaches to implement reversibility of interactions. An implementation and comparison of these approaches could be carried out.

Not in scope of this thesis but nevertheless relevant is the configuration and layout of multiple views next to each other. This configuration could be completed with a feature to save these layouts to and load them from disk respectively.

Chapter 6 lists several performance issues in the handling of `onmousemove` events. The 2.5D treemap spends too much time during picking which is caused by an unnecessary expensive lookup of feature ids. The performance of the geographic visualization suffers on large data sets with many features, when all of them get updated. This could be improved by selectively updating only those features, which have been changed.

Did we achieve our goals?

Is the concept sane in regarding the implementation?

List stuff which was not accomplished in this master thesis

References

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

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

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

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

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

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

Bladh, Thomas, David A. Carr, and Jeremiah Scholl (2004). "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, 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`.

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

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

Carpendale, MST (2003). "Considering visual variables as a basis for information visualisation". In:

Chronic Disease Prevention, National Center for and Health Promotion (2010). *Choropleth Map of Obesity in the United States (1995 to 2008)*. URL: https://homes.cs.washington.edu/~jheer//files/zoo/ex/maps/choropleth.html (visited on 10/09/2017).

Davenport, Thomas H. (Apr. 3, 2013). *How P&G Presents Data to Decision-Makers*. URL: https://hbr.org/2013/04/how-p-and-g-presents-data (visited on 12/04/2017).

Dix, Alan and Geoffrey Ellis (1998). "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, pp. 124–134. DOI: 10.1145/948496.948514. URL: http://doi.acm.org/10.1145/948496.948514.

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

Dukevis (Mar. 27, 2017). URL: http://bl.ocks.org/dukevis/6768900 (visited on 12/04/2017).

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

Figueroa, Pablo, Mark Green, and Benjamin Watson (2001). "A framework for 3D interaction techniques". In: *CAD/Graphics*. Vol. 8, pp. 22–24.

Foster, Mike (2017). *Life as a Modern Day Cartographer - Redefining Cartography for the 21st Century*. URL: http://mjfoster83.github.io/modern-cartography/#/ (visited on 12/13/2017).

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

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

Ho, Quan (May 2013). *Architecture and Applications of a Geovisual Analytics Framework*. Chap. 2. ISBN: 978-91-7519-643-5.

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

Johnson, Brian and Ben Shneiderman (1991). "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, pp. 284–291. ISBN: 0-8186-2245-8. URL: http://dl.acm.org/citation.cfm?id=949607.949654.

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

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

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

Lavrač, Nada et al. (2007). "Data mining and visualization for decision support and modeling of public health-care resources". In: *Journal of Biomedical Informatics* 40.4. 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.

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

Limberger, Daniel et al. (2016). "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, pp. 33–36. ISBN: 978-1-4503-4428-9. DOI: 10.1145/2945292.2945313. URL: http://doi.acm.org/10.1145/2945292.2945313.

MacEachren, Alan M and Menno-Jan Kraak (2001). "Research challenges in geovisualization". In: *Cartography and geographic information science* 28.1, pp. 3–12.

Mann, Robert (Aug. 5, 2016). *Who Lies More: a Comparison*. URL: http://mannmetrics.com/who-lies-more/#more-1 (visited on 12/13/2017).

Mayer, Richard E. (1997). "Multimedia learning: Are we asking the right questions?" In: *Educational Psychologist* 32.1, 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.

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

Nocke, Thomas and Heidrun Schumann (2002). "Meta Data for Visual Data Mining". In: *Proceedings Computer Graphics and Imaging, CGIM'02*. Citeseer.

Observatory of Economic Complexity (n.d.).

Peltier, Jon (Aug. 4, 2016). *Diverging Stacked Bar Charts*. URL: https://peltiertech.com/diverging-stacked-bar-charts/ (visited on 12/13/2017).

Poleto, Thiago, Victor Diogho Heuer de Carvalho, and Ana Paula Cabral Seixas Costa (2015). "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, 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.

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

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

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

Rose, Margaret (July 1, 2017). *Guide to telling stories with data: How to share analytics insights*. URL: http://searchbusinessanalytics.techtarget.com/definition/data-visualization (visited on 12/04/2017).

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

– (1996). "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, pp. 336–. ISBN: 0-8186-7508-X. URL: http://dl.acm.org/citation.cfm?id=832277.834354.

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

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

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

Declaration of Authorship

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Master Thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Potsdam, December 14, 2017

Robert Schäfer