

Master's Thesis

Multiple Coordinated Views on Massive Geo Data

Multiple koordinierte Visualisierungen großer Mengen an Geodaten

Robert Schäfer

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

Hasso Plattner Institute for Digital Engineering
Department of Computer Science



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

Supervisors:

Prof. Dr. Jürgen Döllner
Benjamin Hagedorn
Jan Klimke

Hasso Plattner Institute
Potsdam, Germany
November 28, 2017

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

Contents

List of Figures	v
List of Tables	viii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Hypothesis	2
1.4 Contributions	3
1.5 Structure of the Work	3
2 Related Work	5
2.1 Foundations	5
2.1.1 Data Visualizations	5
2.1.2 Visual Variables	6
2.1.3 Treemaps	6
2.1.4 Geographical Data Visualization	8
2.1.5 Coordinated Multiple Views	11
2.2 Libraries, Frameworks and Standards	11
2.3 Interaction Theory	12
2.3.1 Interaction Categories	12
2.3.2 Interaction Models	13
3 Analysis	17
3.1 Single Visualization Interactions	17
3.2 Multiple View Interactions	24
3.3 Requirements	25
3.4 Existing Interactions	25
3.5 Comparison of client-side component frameworks	27
4 Concept	31

4.1	Terminology	31
4.2	Formalization of an interaction	33
4.3	Conceptual Framework	33
4.4	Shared data model.....	34
4.5	Predefined Interaction Functions	35
4.6	Communication pattern	38
5	Implementation.....	41
5.1	Architecture	41
5.2	Software patterns.....	42
5.3	Geographical visualization with React and Leaflet	44
5.3.1	GeoJSON component	45
5.4	MultiviewController	47
5.5	Implemented interactions	50
6	Evaluation	53
6.1	Evaluation of Use Cases	53
6.1.1	Explain outliers with a geographical context	53
6.1.2	Find an unusually cheap gas station in the center of Berlin	54
6.1.3	Explain influx of sparsely populated administrative districts	54
6.1.4	Large districts by area with a high unemployment rate ..	56
6.2	Performance Evaluation	56
6.3	Evaluation of Requirements	60
7	Conclusion and Future Work	63

List of Figures

2.1	Bertin's [5] original visual variables.	7
2.2	First hierarchy level of German exports	7
2.3	Second hierarchy level of German exports	8
2.4	Example of a 2.5D tree map	9
2.5	Charles Minard's map of Napoleon's disastrous Russian campaign of 1812. Minard managed to represent six values in two graphical dimensions: The number of Napoleon's troops, the travelled direction and distance, latitude and longitude relative to specific dates and the temperature.	9
2.6	Choropleth Map of Obesity in the United States in 2008. Present of population classified as "obese" (Body Mass Index in excess of 30), by state.	10
2.7	Airline on-time performance: Correlation of time of day with arrival delay. Most recent flight with a delay of more than 100 minutes selected.	11
3.1	Line graphs are used to display trends	18
3.2	A multi set bar charts is a variation of a bar chart	18
3.3	A histogram is a bar chart over a continuous interval	19
3.4	Bubble charts and scatter plots are similar regarding interactions	20
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 ..	21
3.6	A population pyramid can be modeled as a stacked bar chart ...	21
3.7	Tree maps and sunburst diagrams are ideal to show hierarchies ..	22
3.8	Choropleth maps focus on a density while flow maps show a migration of data	23
3.9	Similar to a calendar, a gantt chart shows activities and the progress along a time line	24

3.10	The screenshot-tool “shutter” shows a magnified detail view of the area around the mouse cursor in the lower right corner of the screen	24
3.11	Items can be highlighted with a click, Bavaria is currently highlighted. A mouse over reveals a tooltip showing item properties. The menu on the left side allows to change the data set and the specific base visualization.	26
3.12	Only gas stations with a price for E10 within 1.349 Euro and 1.589 Euro are displayed in the map	27
3.13	Comparison of frontend frameworks.....	29
4.1	A data structure for tabular, hierarchical and relational data ...	34
4.2	The message flow of the communication pattern: Every view can decide for itself whether to trigger an interaction or respond to an interaction with a visual effect.	38
5.1	Architecture of the reference implementation.	42
5.2	The sequence diagram shows the notification of different components. The user first chooses a feature set and hovers over a polygon in the geographical map.	43
5.3	Implementation of the component pattern in EmberJS. The example shows a page of a webshop. The customer is about to order the items in the shopping cart.	44
5.4	User distribution of RUNDNFUNK MITBESTIMMEN across German federal states	50
6.1	The 2.5D tree map shows gas stations with a layout based on the brand name and height and colour mapped to price. Relatively expensive gas stations, with respect to the other gas stations of that brand, are often located next to highways.	54
6.2	A multi select allows to collect a group of interesting features. ...	55
6.3	Three unsuspicious districts happen to be located next to Munich. Apparently people move out of Munich to avoid high rents.	55
6.4	The layout of the 2.5D tree map is based on area of the district, the colour based on the unemployment rate. We did a multi select of all rural districts with a high unemployment rate. We can see a strong correlation with a location in the east of Germany.	56
6.5	During profiling of the “Immoscout” data set, only 16 frames per second get rendered	58

6.6	Visualizing just the 2.5D tree map, almost the entire CPU time is spent during Scripting	58
6.7	Every highlighting interaction comes with a spike of CPU time for rendering and painting. Focusing on a feature redraws the background and leads to an increase of painting time.	58
6.8	The geographical visualization spends much more time during re-rendering and painting.	58
6.9	Event handler <code>handleHighlight</code> takes about 90ms to finish. During this time, the list of features is iterated and <code>setStyle</code> executed.	59
6.10	The number of features is crucial for the time spent during a highlighting interaction.	59
6.11	Data set “Landkreise” shows less time spent on painting and rendering.	59
6.12	A timeline without any significant drop of frames per second is achieved with the geographical visualization without 2.5D tree map of the “Landkreise” data set.	60
6.13	Without a 2.5D tree map the CPU is idling for almost 50% of the time.	60

List of Tables

3.1	Interactions for line charts	18
3.2	Interactions for bar charts	19
3.3	Interactions for bubble charts	20
3.4	Interactions for stacked bar charts	21
3.5	Interactions for hierarchical visualizations	22
3.6	Interactions for geographical visualizations.....	23
3.7	Interactions for temporal visualizations.....	24
6.1	Hardware specifications	57
6.2	Data sets used for performance profiling.....	57

1

Introduction

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

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

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

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

1.1 Motivation

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

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

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

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

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

1.2 Problem Statement

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

1.3 Hypothesis

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

1.4 Contributions

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

1. Concept

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

2. Implementation

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

3. Integration of 2.5D tree map and 2D map

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

4. Demonstration and evaluation

Coordinated multiple view layouts and suitable views are implemented and tested for the selected test data. Based on the test data sets, the coordinated multiple view implementation is examined and evaluated for design criteria [43], general usability aspects [38] and usage for typical visual analytics tasks.

1.5 Structure of the Work

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

implementation. Finally sum up the main contribution in Section 7 and outline the future work.

2

Related Work

no section without text

2.1 Foundations

no section without text

2.1.1 Data Visualizations

Data visualizations is a means of visual communication and have steadily developed since the 16th century [19]. Otherwise abstract information is visually represented, making complex data more accessible, understandable and usable. Kusinitz [29] explains that the human brain processes visual information 60,000 times faster than text and visual content makes up even 93% of all human communication. The purpose of data visualizations is twofold, according to the Interaction Design Foundation: sense-making and communication. Statistical information is abstract and in data visualization “we must find a way to give form to that which has none.” [17] Successful data visualizations helps the human user to derive knowledge and meta data from the visualization itself, Nocke and Schumann [35] call it “visual data mining”.

Data-driven decision support systems are applications to support businesses and organizational decision-making activities in which data visualizations play a key part [30] [36]. Visualizations are an obvious choice for managers who demand a quick overview on performance data. Stephen Few’s book “Show me the numbers” was named after the phrase often used by sales managers who can’t afford to wade through lengthy reports.

Prof. Döllner mentioned the emerging “cockpit” views

We can expect to see these technologies more and more in business applications. McAfee and Brynjolfsson [34] from the MIT Center of Digital Business showed that organizations driven most by data-based decision making had 4% higher productivity rates and 6% higher profits. However, little research has been done regarding the performance of coordinated multiple views in the field of decision making. There might be a great potential. Back in 1997 Mayer [33] conducted eight studies to compare the effect of using multimedia on university students. The studies showed that when using combined visual and verbal explanations the generation of creative problem solutions increased by an average of more than 50%.

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

2.1.2 Visual Variables

French cartographer Jacques Bertin introduced seven visual variables in 1967 [5]. We can see an example for each in Figure 2.1. These visual variables are used in cartography but can also be applied to data visualization in general. Carpendale [11] explains in detail their use in terms of computational information instead of printed cartography. Garlandini and Fabrikant [20] put these visual variables under systematical validation procedures. The authors conclude that the variable size provides the most accurate and efficient performance while the variable orientation provides the least performance.

2.1.3 Treemaps

The visualization of hierarchical data has a long tradition. The traditional visual representation of a tree is a directed graph with the root node at the top. An everyday use case is a directory tree example of a file system, e.g. in file browsers or command line utilities like `tree` in UNIX based operating systems. As Shneiderman [41] mentions, this visualization becomes increasingly large when displaying

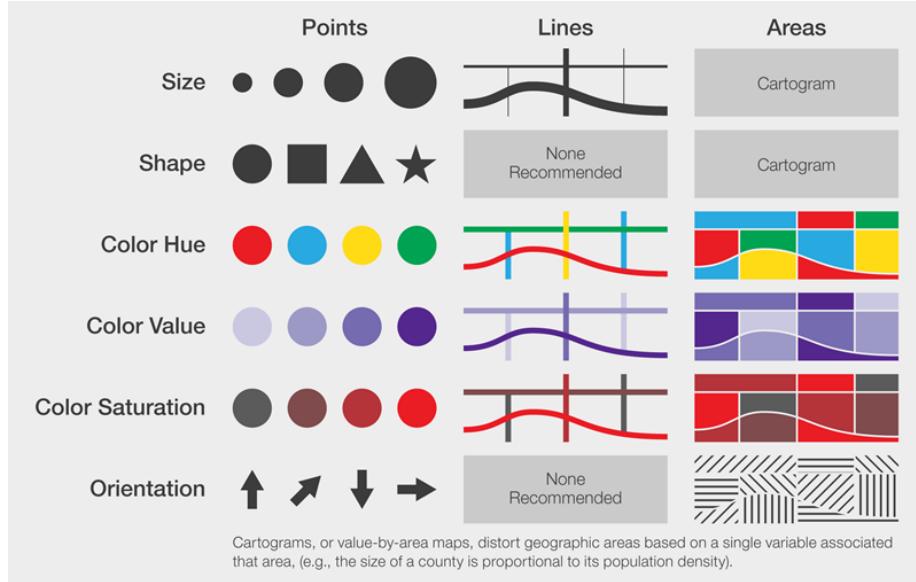


Fig. 2.1: Bertin's [5] original visual variables.

more than one node and soon exceeds the entire screen size. Johnson and Shneiderman [26] proposes the tree map visualization technique, in which each node is a rectangle whose area is proportional to a specified dimension. Rectangles contain sub-branches of the node as tiles, thereby expressing hierarchy information.

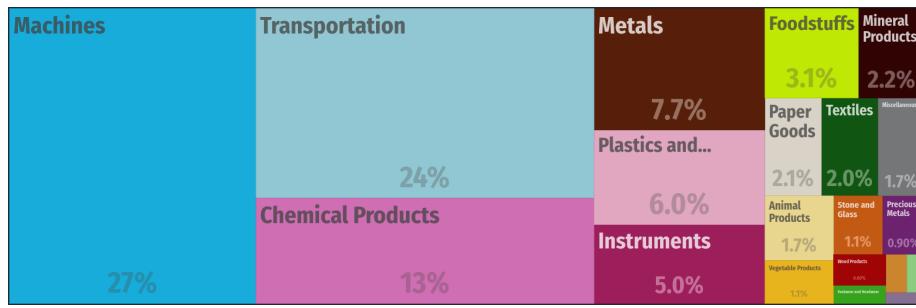


Fig. 2.2: First hierarchy level of German exports

Why is the layout different in the second image?

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

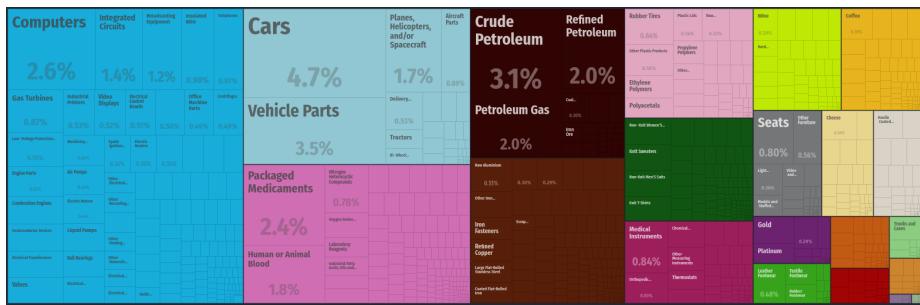


Fig. 2.3: Second hierarchy level of German exports

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

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

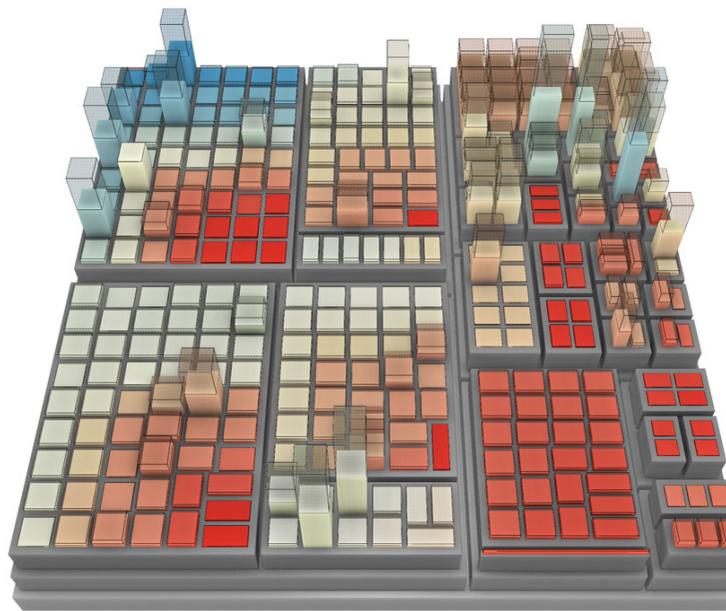
3D tree maps is a concept introduced by Bladh, Carr, and Scholl [8] in 2004. The authors transfer the concept of tree maps from two dimensional into three dimensional space. They introduce StepTree [8], which is a three dimensional tree map to display a directory layout. It “differs from Treemap in that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

2.5D tree map is a term coined by Limberger et al. [32]. A 2.5D tree map emphasizes physical constraints of a 3D tree map, i.e. a 2.5D tree map has all items attached to the ground. We can see an example of a 2.5D tree map in figure 2.4

2.1.4 Geographical Data Visualization

introduction

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



© Hasso-Plattner-Institut[15]

Fig. 2.4: Example of a 2.5D tree map

early example of a flow map is Charles Minard's depiction of Napoleon's ill-fated march on Moscow in 1812 [13].

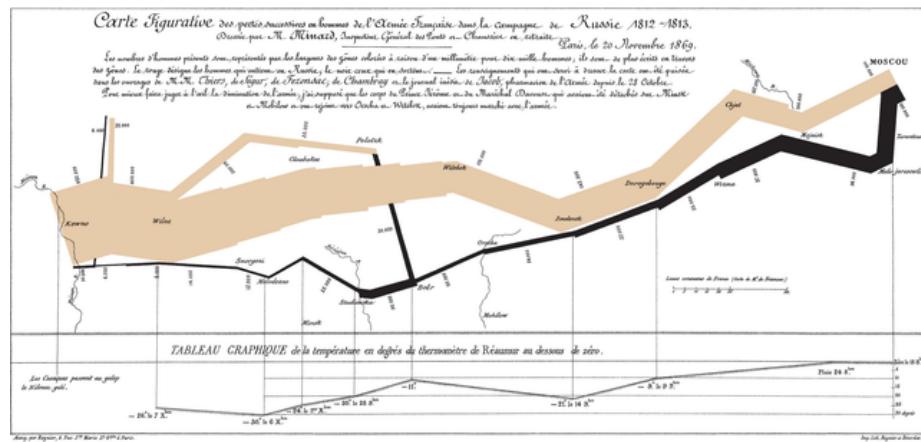
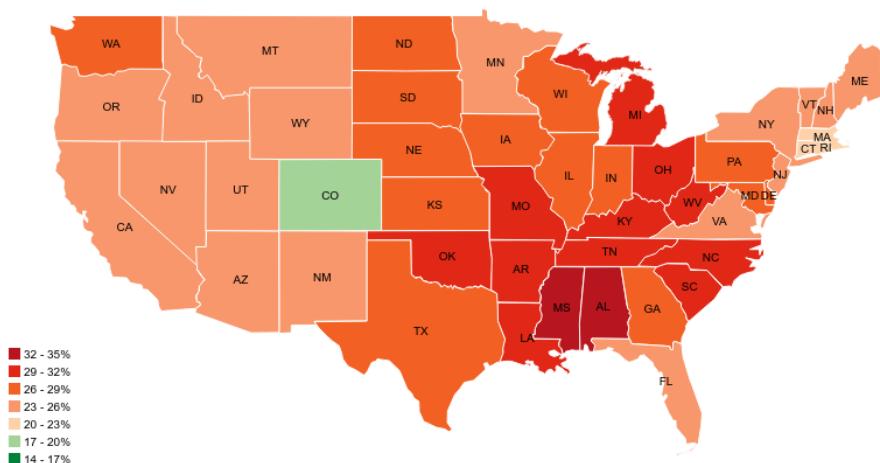


Fig. 2.5: Charles Minard's map of Napoleon's disastrous Russian campaign of 1812. Minard managed to represent six values in two graphical dimensions: The number of Napoleon's troops, the travelled direction and distance, latitude and longitude relative to specific dates and the temperature.

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



National Center for Chronic Disease Prevention and Health Promotion [12]

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

Graduated Symbol Maps is a good alternative to a choropleth maps, as it alleviates one of its downsides: Larger areas may appear more emphasized than smaller ones, thus confounding geographic area with data values. Instead, symbols are placed over the regions of a map. As Heer, Bostock, and Ogievetsky [22] point out, these symbols allow for more visual variables to be visualized, i.e. symbol size, shape and colour.

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

2.1.5 Coordinated Multiple Views

According to Roberts [38] coordinated multiple views is just “a specific exploratory visualization technique that enables users to explore their data”. coordinated multiple views are characterized by the fact, that they show multiple views side-by-side. We can see an example in Figure 2.7. It displays an on-time performance of airlines, visualized with the “Crossfilter” javascript library. The user can set the borders of an interval with the mouse in each of the views. The visualization takes the most recent 80 flights from the database that match all given filters. All visualizations are then updated in real time. As we can see in the example in Figure 2.7 there seems to be a correlation of a long delay with a later time of the day.

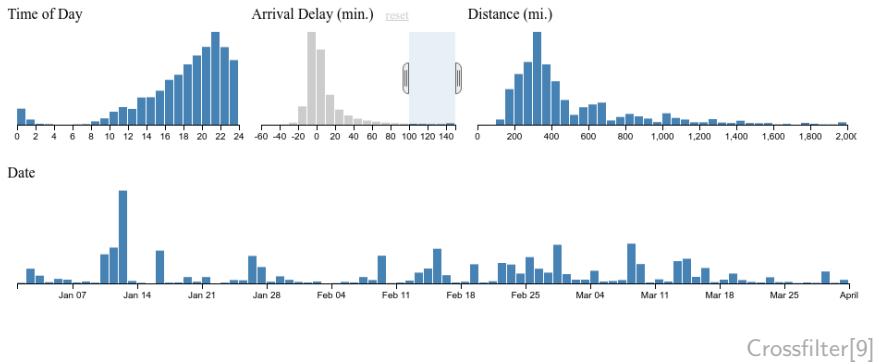


Fig. 2.7: Airline on-time performance: Correlation of time of day with arrival delay. Most recent flight with a delay of more than 100 minutes selected.

Brushing and Linking Most multiple coordinated views also provide some kind of brushing technique. “The technique of brushing is the principle approach, where elements are selected (and highlighted) in one display, concurrently the same information in any other linked display is also highlighted.”[38]

2.2 Libraries, Frameworks and Standards

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

PubSubJS is a topic-based publish/subscribe library written in JavaScript [39]. Topics can be registered hierarchically, with subtopics delimited by dots. A

subscription to topic `mcv.select.focus` will be notified only for `focus` interactions whereas a subscription of `mcv.select` will be notified for both `focus` and `highlight` interactions. Furthermore, topics are published asynchronously, so if the user interacts with a visualization, that does not block code execution.

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

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

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

2.3 Interaction Theory

According to Ho [23] interactions are a crucial part of data visualizations, yet most research in the area of data visualization still focuses on visual representations. Roughly speaking, research on interaction falls into these groups: How to categorize interaction techniques? How to find new interaction techniques and apply those to visualizations?

Give a rough overview of this section

2.3.1 Interaction Categories

Shneiderman [40] classifies interactions into these groups: (1) Gain an *overview* of the entire collection, (2) *zoom* in on items of interest, (3) select an item or group and get *details* when needed, (4) view *relationships* among items, (5) keep a *history* of actions to support undo, (6) allow *extraction* of sub-collections and of the query parameters.

In 1997 Shneiderman [40] 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 [14] identified these categories: (1) *Highlight and focus* particular subsets of the data, (2) instead of displaying everything simultaneously *access extra information* by drilling down the data, (3) zoom in and out to give an *overview and context*, (4) *change parameters* of the *same representation*, e.g. another baseline of a stacked bar char, (5) *change representation* of the *same data* by switching the chart type, (6) *link representations* to determine the relationship between items.

In 2002, Keim [28] comes up with the following classification: (1) Dynamic *projection* to show all combination of data attributes mapped to the axis of a diagram, (2) focus on a smaller subsets by *filtering* out parts of the data, (3) *zoom* into a subset of the data and get a higher level of detail, (4) preserve an overview of the data during drill-down operations is called *distortion* (5) and finally *link and brush* visualizations, to highlight the same data points in multiple visualizations.

The most recent classification was done in 2007 by Yi, Kang, and Stasko [45] listing seven categories: (1) *Select* to mark something as interesting, (2) *explore* to show something else, (3) *reconfigure* to show a different arrangement, (4) *encode* to show a different representation, (5) *abstract/elaborate* show more or less detail, (6) *filter* show something conditionally, (7) *connect* show related items.

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

Give one example for each category of the chosen classification

2.3.2 Interaction Models

no section without text

Space-Time Cube Operations is a concept introduced in 2014 by Bach et al. [4] to map temporal data into two dimensional visualizations. Space-time-cubes are used to model two attributes of continuous data with temporal data along a third axis, therefore the name *cube*. While the transformations are rather static it is also possible to introduce activity into the transformations. The authors describe user-independent *animations* and user-controlled *interactions*. E.g. a transformation may display a given slice of the cube. An animation would display one slice at a time and display the next slice every second. Whereas the interaction would show the slice determined by a user-controlled slider. Various

transformations and their best use in practice are evaluated in this work. The work focuses on temporal data and otherwise continuous data. Interactions are not seen as an abstract entity, that need to be agnostic of the underlying data structure and visualization. The authors admit “our framework does not provide much guidance for interaction design: the design space for interactive operations has only been partially explored.”[4, Other limitations, p. 15]

ITlib[18] 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”[18, Basic concept, p. 2]. These filters are the smallest process unit in the data flow. Composed of input and output ports, they communicate with other filters, to receive data input from predecessors and send data output to successors. The framework specifies and stores the interaction techniques along with its filters, the execution model and the scene in XML documents. The authors chose XML because it can be parsed easily and they generate code in order to target various virtual reality toolkits and environments.

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

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

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

$$IV([D], V, I) \quad (2.1)$$

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

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

$$IV'(IV, V', I') \quad (2.2)$$

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

3

Analysis

Sections 3.1 and 3.2 abstract and deduce the essential characteristics of any interaction. These characteristics are relevant for a formalized language of coordinated multiple views. To accomplish this goal, the approach is to deduce the characteristics by a list of examples. What are the expected data structures for each visualization? What visual variables can be used to show an interaction? A list of possible interactions is given for each visualization. Those interactions will be classified according to Yi, Kang, and Stasko [45] and the relevant subject of the interaction is specified.

Section 3.3 gives a set of requirements which can be used to evaluate a framework of coordinated multiple views.

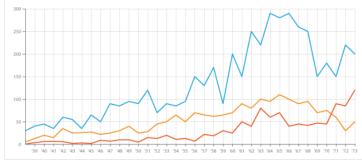
Finally in Section 3.5 existing tools are evaluated with respect to their suitability for coordinated multiple views. In order to accomplish that, advantages and disadvantages of libraries and frameworks are compared and a final decision is made.

3.1 Single Visualization Interactions

The data visualization catalogue by Severino Ribecca list many of the most used data visualizations[37]. This section covers a list of common data visualizations from that catalogue as examples. The expected data structure and a list of possible interactions is systematically analysed.

Line graphs

Line graphs display how quantitative values have changed over time. They are perfectly suited to show trends or compare multiple series of data with each



(a) Line graphs

Fig. 3.1: Line graphs are used to display trends

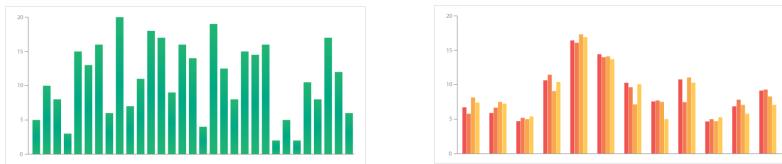
other. The expected data format is *tabular*, since one or many series of data in parallel are visualized.

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 nature of the visualization. However, an interaction with the line graph can alter the (1) shape (2) color or (3) size of lines to create a visual effect. 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 3.1 shows a list of conceivable interactions in a line graph.

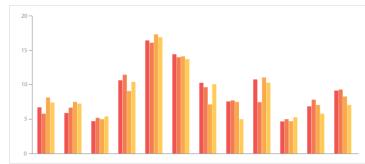
Table 3.1: Interactions for line charts

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

Bar charts and 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

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.

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 3.2 list some possible interactions.

Table 3.2: Interactions for bar charts

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

Histograms

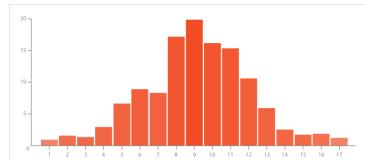


Fig. 3.3: A histogram is a bar chart over a continuous interval

Histograms visualise the distribution of data over a continuous interval or certain time period. A special type is the population pyramid, which is a pair of back-to-back histograms, one for each sex. Histograms and bar charts expect the same kind of data, i.e. a *tabular* format. Almost the same interactions as in Table 3.2 can be applied to histograms. Except a re-ordering of bars along the x-axis, because the histogram constrains the position of bars along the interval.

Bubble charts and scatter plots

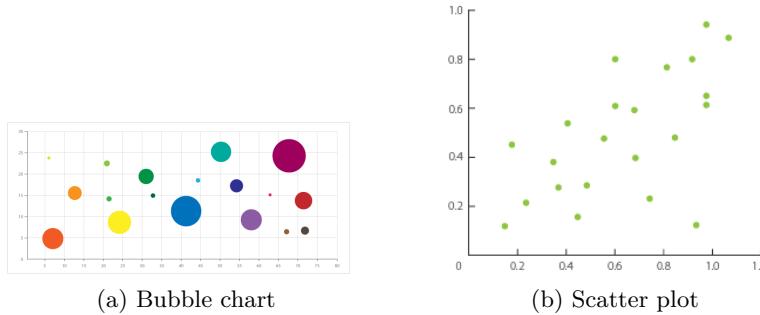


Fig. 3.4: Bubble charts and scatter plots are similar regarding interactions

Both bubble charts and scatter plots are popular choices to visualize variables from two values. Points are placed with the two values in Cartesian coordinates in order to detect relationships and correlations. In case of bubble charts, each point is displayed as a bubble with a third value encoded in the size of the bubble. 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 3.3, interactions also include a zooming and movement of the viewpoint.

Table 3.3: Interactions for bubble charts

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

Stacked bar charts

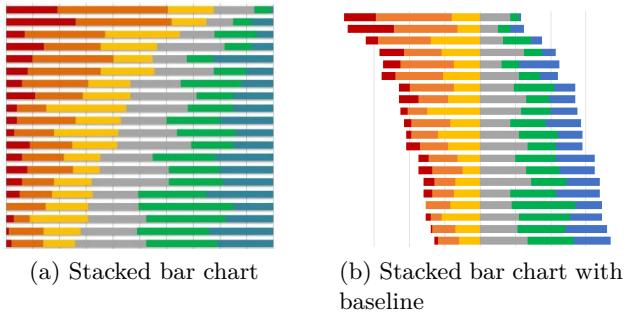


Fig. 3.5: Stacked bar charts can be ordered along a baseline or stretch to 100% width to show the percentage-of-the-whole of each group

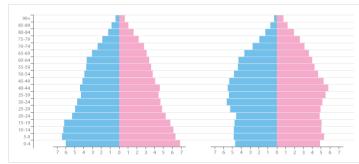


Fig. 3.6: A population pyramid can be modeled as a stacked bar chart

Unlike a multi-set bar graph which displays their bars side-by-side, stacked bar graphs segment their bars of multiple datasets on top of each other. A baseline, as shown in figure 3.5 might be modeled as two back-to-back multi-set bar graphs. A reordering would e.g. move one data set from the left side to the right side. A stacked bar chart also expects *tabular* data. If the stacked bar chart has a baseline, often the sign of the numeric value defines the placement of the segment on the left or on the right side. Table 3.4 shows possible interactions, including the highlighting of a feature, a change of color mapping or a reordering of the baseline.

Table 3.4: Interactions for stacked bar charts

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

Hierarchical visualizations

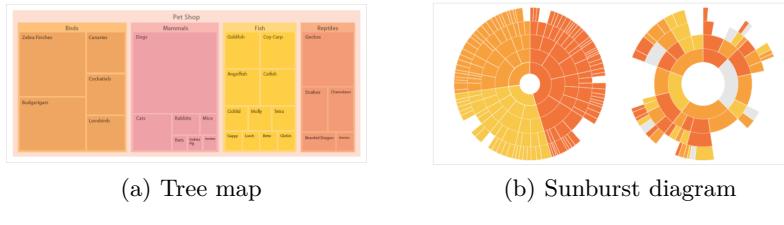


Fig. 3.7: Tree maps and sunburst diagrams are ideal to show hierarchies

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

Therefore, both tree map and ring diagram expects *hierarchical* data in form of a tree. Each node needs to have at least one quantitative value for layouting, additionally, each node may be assigned a color.

As we are describing hierarchies, the maximal depth of tree may be increased or decreased. Again, interactions could include a highlighting of features and a change of color encoding. Both visualizations may show only a subtree. E.g. a click on a box in the treemap opens another treemap focused on the subtree. Similarly a click on a slice of the ring would surround the most external ring with the children of the feature. Table 3.5 gives a more comprehensive list of interactions.

Table 3.5: Interactions for hierarchical visualizations

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

Geographical Data

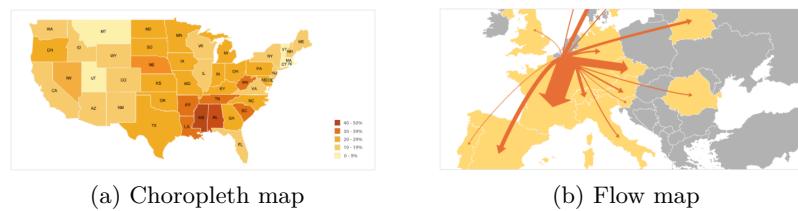


Fig. 3.8: Choropleth maps focus on a density while flow maps show a migration of data

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

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

Table 3.6: Interactions for geographical visualizations

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

Activity diagrams

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

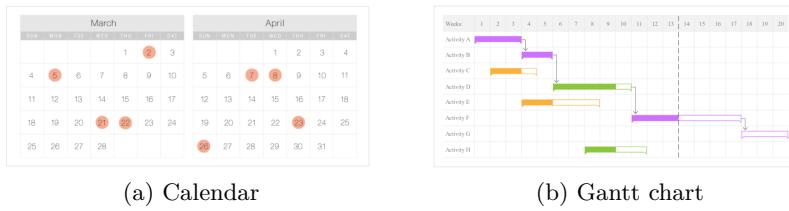


Fig. 3.9: Similar to a calendar, a gantt chart shows activities and the progress along a time line

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.

Table 3.7: Interactions for temporal visualizations

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

3.2 Multiple View Interactions

Detail view

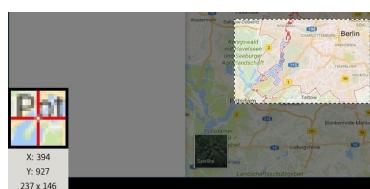


Fig. 3.10: The screenshot-tool “shutter” shows a magnified detail view of the area around the mouse cursor in the lower right corner of the screen

3.3 Requirements

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

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

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

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

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

3.4 Existing Interactions

We will now classify the implemented interactions of our two applications according to the classification by Yi, Kang, and Stasko [45] in Section 2.3.1.

In our visual analytics platform possible interactions can be categorized into the classes *select*, *explore*, *reconfigure*, *encode* and *filter*. As seen in Figure 3.11

the user can *select* one item in the view by clicking on it. The user can reveal a tooltip showing the item properties by hovering with the mouse on the item, which is another *selection*. The user can *explore* the map in the usual manner: If the user drags with the mouse on the map, a panning operation is performed with the viewpoint focused on Germany, i.e. the camera moves around like a turntable. The zoom factor can be changed by scrolling on the canvas of the map. *Encode* and *reconfigure* techniques are performed through the menu on the left side: Under the “features” tab, the user can *reconfigure* different data sets and the displayed diagram, e.g. a tree map visualization based on the geometry shape, cubes or voronoi regions. The tab “Dimensions” allows the user to *encode* properties of a data set to visual attributes, e.g. the height, color and texture of an item. The tab “Filter” can be used to reduce the displayed data set along a range of continuous values. Figure 3.12 shows the range of visible values in the left menu. When the user drags the slider, the items in the map on the right side are updated interactively.

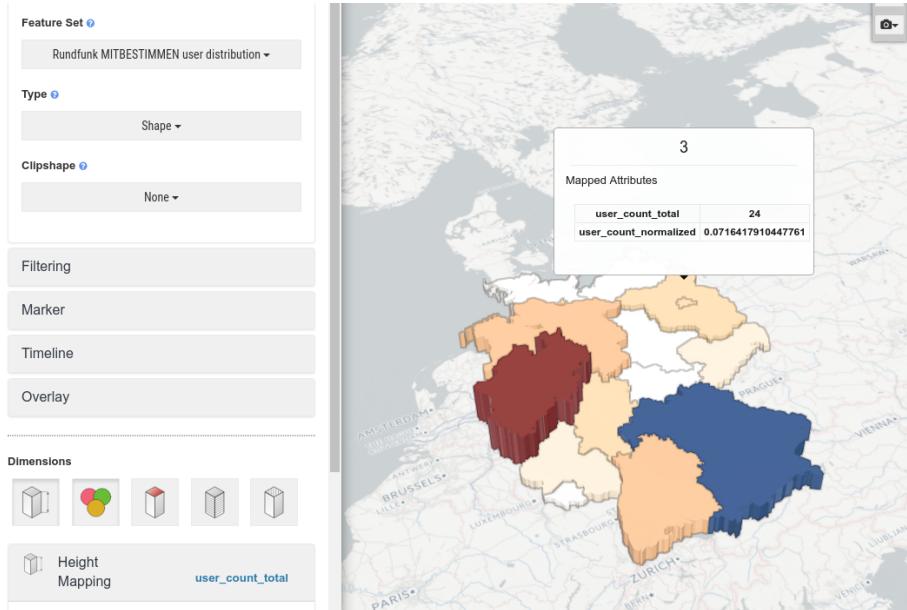


Fig. 3.11: Items can be highlighted with a click, Bavaria is currently highlighted. A mouse over reveals a tooltip showing item properties. The menu on the left side allows to change the data set and the specific base visualization.

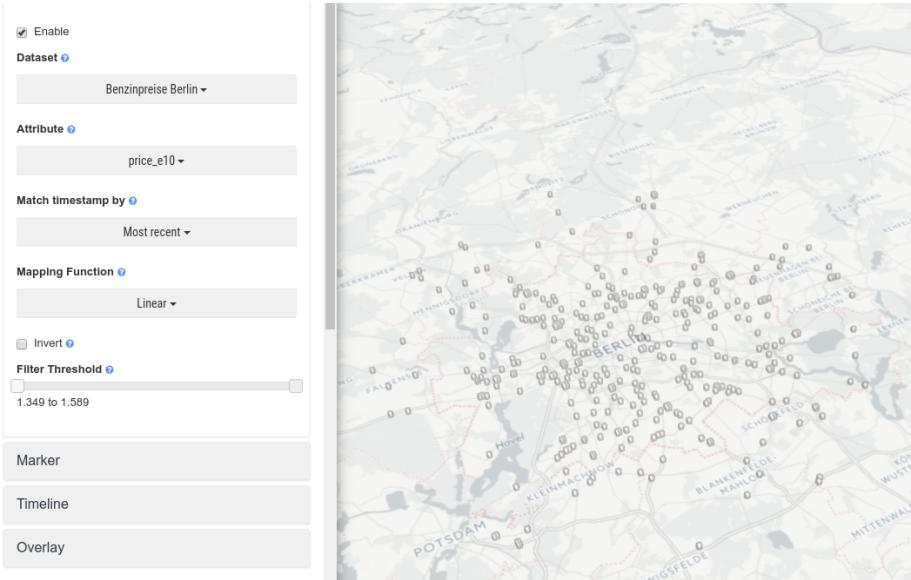


Fig. 3.12: Only gas stations with a price for E10 within 1.349 Euro and 1.589 Euro are displayed in the map

3.5 Comparison of client-side component frameworks

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

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

Web components is a recent standard of the W3C[2] to bring component-based software engineering to the world wide web. Web components are a set of web platform APIs to create new custom, reusable, encapsulated HTML tags that can be used in web pages and web applications [31]. 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 engine of EmberJS[44]. 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[27], 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 [25]. With 18,469 stars on Github it is the most popular framework for web components at the time of writing. Polymer has a large community and comprehensive documentation and therefore more suitable than GlimmerJS to build coordinated multiple views.

But critically, coordinated multiple views require a means to exchange data between views, which is specific to interactions. The web component specification, unfortunately, does not specify how arbitrary JavaScript objects can be passed to web components. String based attributes are supported, as seen in Listing 3.1. To pass rich data to components however, web component frameworks have to roll their own data flow and syntax. Google Polymer's syntax to pass rich data to a components is shown in Listing 3.2. But this is a proprietary solution that abandons standard HTML.

```

1 <google-map fit-to-marker api-key="AIzaSyD3E1D9b-
    Z7ekrT3tbhl_dy8DCXuIuDRC">
2   <google-map-marker latitude="37.78" longitude="-122.4"
        draggable="true"></google-map-marker>
3 </google-map>
```

Listing 3.1: An example of string based attributes of web components [6]

```

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

Listing 3.2: An small syntax example how Google Polymer passes rich data to a component

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

ReactJS is a JavaScript library for building user interfaces[24]. React is explicitly not implementing web components and is not going to implement web components in the future. It has, in return, a well-known way of integrating the component framework into a legacy application built with e.g. jQuery. Along

with its major advantage of easy integration, it has a striving community, lots of documentation and tutorials and it is well tested.

As a summary, if there is (1) no obligation to implement web components (2) and an easy integration into an existing application is necessary, then React is the perfect choice for coordinated multiple views. Figure 3.13 shows the pros and cons of each framework for the use case.

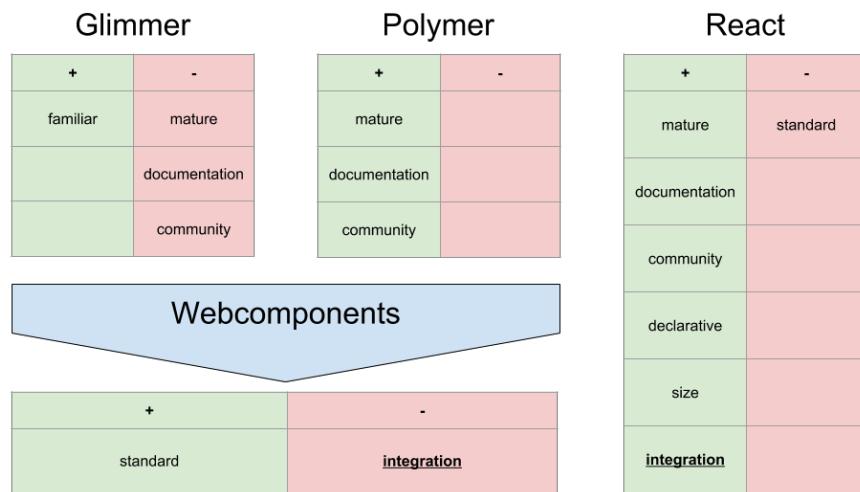


Fig. 3.13: Comparison of frontend frameworks

4

Concept

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

4.1 Terminology

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

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

Effect is the change of the visual representation subsequent to the interaction. In order to be perceivable by the user, the interaction must have some visual effect, i.e. a change in the visual representation of the data. Examples are: A change of colour of a selected bubble, a movement of the viewpoint, a rearrangement of attributes in a parallel plot or a higher level of detail in a 2.5D tree map. Every view is responsible of its visual *effects*. Those are implementation details

as well and they are not shared by other views. Obviously visual effects are not shared, as it depends on the visualization if visual variables are constrained or available to express a visual effect.

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

$$I(T, E) \quad (4.1)$$

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

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

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

Interaction function is the declaration and definition how the subject of the interaction should be changed. The aforementioned explicit meaning of the interaction is the smallest unit of information of the interaction. This meaning is shared among multiple views, it is agnostic of how the interaction is implemented. Therefore the meaning is shared between multiple views. We call this meaning the *interaction type*. Some examples of interaction types include: Selection, Deletion, Point-of-Interest, Filtering, Reordering, Re-encoding. Interaction types can be classified with the interaction categories of Yi, Kang, and Stasko [45].

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

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

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

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

$$S(V, F(IS)) \quad (4.2)$$

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

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

4.2 Formalization of an interaction

We formalize an interaction I as follows:

$$I(T, S(V, F(IS)), E) \quad (4.3)$$

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

4.3 Conceptual Framework

no section without text

4.4 Shared data model

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

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

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

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

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

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

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

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

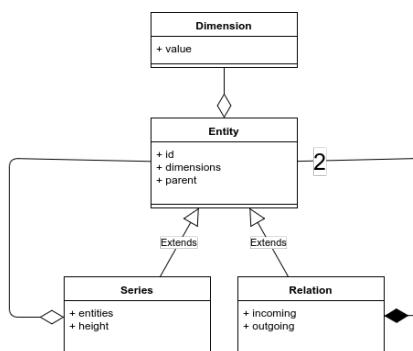


Fig. 4.1: A data structure for tabular, hierarchical and relational data

4.5 Predefined Interaction Functions

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

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

$$\mathbb{E} : \mathbb{E} \subseteq \mathbb{N} \quad (4.4)$$

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

$$\mathbb{D} : \mathbb{D} \subseteq \Sigma^* \quad (4.5)$$

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

$$Space(\mathbb{D}) \quad (4.6)$$

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

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

$$\mathbb{V} = \{x, y, y2, y3 \dots yn, z, height, colour, size, shape, orientation \dots\} \quad (4.7)$$

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

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

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

$$Select : \emptyset \rightarrow \mathcal{P}(E) \quad (4.8)$$

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

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

$$Filter : E \rightarrow \{\perp, \top\} \quad (4.9)$$

The *Filter* function describes which entities are part of the visualization. It can be implemented explicitly or implicitly. An explicit implementation would return `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.

$$Window : \mathbb{D} \rightarrow \mathcal{P}(Space(\mathbb{D})) \quad (4.10)$$

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

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

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

$$Order : \mathbb{E} \times \mathbb{E} \rightarrow \mathbb{R} \quad (4.11)$$

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

Similar to *Filter*, the *Order* function can either be implemented explicitly or implicitly. An explicit implementation returns a value r based on the relative position of $e1$ to $e2$ 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.

$$Encode : \mathbb{D} \rightarrow \mathbb{V} \quad (4.12)$$

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

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

A user clicks on a bar in a bar chart and this feature then changes its background colour. To coordinate the highlighting, the bar chart view will formulate a new

sentence composed of a *Select* function and the verb *highlight*. The function returns the set with the highlighted entity.

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

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

4.6 Communication pattern

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

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

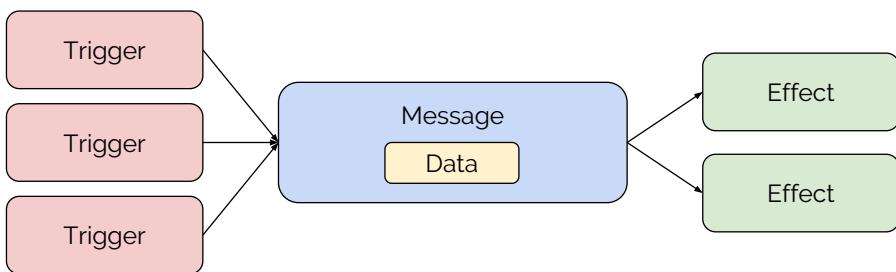


Fig. 4.2: The message flow of the communication pattern: Every view can decide for itself whether to trigger an interaction or respond to an interaction with a visual effect.

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

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

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

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

Implementation

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

5.1 Architecture

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

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

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

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

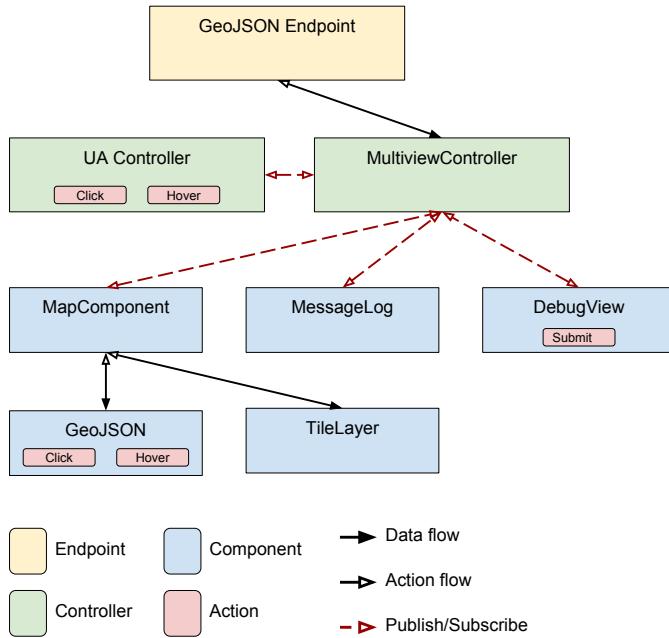


Fig. 5.1: Architecture of the reference implementation.

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

5.2 Software patterns

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

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

Publisher subscriber In our particular case we apply a special form of the observer pattern, the so called “Publish-subscribe” pattern[16]. Publish-subscribe is a messaging pattern which is widely used in message queues. In this scenario, senders of messages simply categorize their messages which will be consumed by subscribers of the category. The scenario has very low coupling, publishers do not even need to know the existence of subscribers.

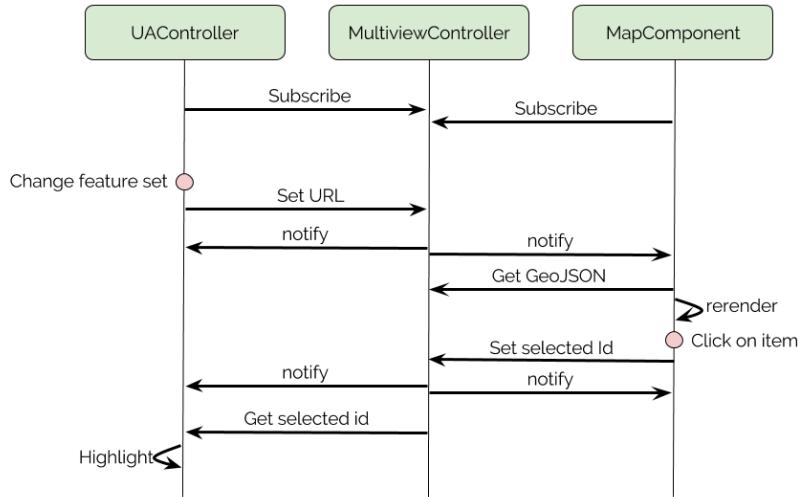


Fig. 5.2: The sequence diagram shows the notification of different components. The user first chooses a feature set and hovers over a polygon in the geographical map.

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

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

Actions up — Data down

Version 2.0 of Ember introduced a common phrase how to use this pattern effectively: “Data down, actions up”[42] In the domain of coordinated multiple views actions would mean user interactions, e.g. a click on a feature. The action will notify the controlling coordinated multiple view component. Actions may change data, and the changes will be passed to all dependent views. These views are then rerendered.

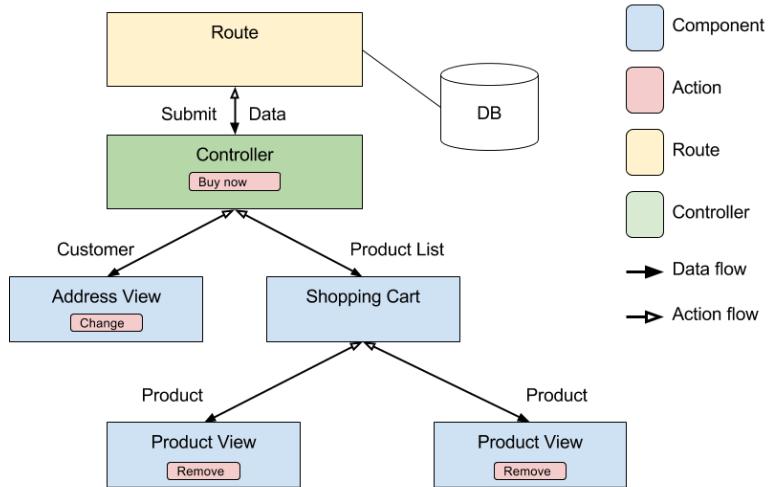


Fig. 5.3: Implementation of the component pattern in EmberJS. The example shows a page of a webshop. The customer is about to order the items in the shopping cart.

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

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

5.3 Geographical visualization with React and Leaflet

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

```

1 render() {
2   return (
3     <div className="multiview-map-component">
4       <Map ref={(m) => this._map= m} bounds={this.bounds()}>
5         <TileLayer
6           attribution='&copy; <a href="http://osm.org/copyright">
7             OpenStreetMap</a> contributors'
8         </TileLayer>
9       </Map>
10      <div>
11        <h3>Map</h3>
12        <p>This is a map component</p>
13      </div>
14    </div>
15  )
16}

```

```

7   url='http://{s}.tile.osm.org/{z}/{x}/{y}.png'
8   />
9   { this.state.geojsonUrl && this.state.geojson &&
10    <GeoJSON
11      key={this.state.geojsonUrl}
12      data={this.state.geojson}
13      style={this.featureStyle}
14      pointToLayer={this.pointToLayer}
15      onEachFeature={this.onEachFeature}
16    >
17    </GeoJSON>
18  }
19  </Map>
20  </div>
21 );
22 }

```

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

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

5.3.1 GeoJSON component

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

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

click or a mouse move on each polygon. Listing 5.2 shows the event handlers added to the map.

```

1 onEachFeature(feature: geojson.Feature<geojson.GeometryObject>,
    layer: Leaflet.Layer){
2     this.state.layerList.push(layer);
3     layer.on({
4         mouseover: () => {
5             this.state.controller.publish('mcv.select.highlight', [
6                 Number(feature.id)]);
7         },
8         click: (event: Leaflet.LeafletMouseEvent) => {
9             if(event.originalEvent.ctrlKey){
10                 this.state.controller.publish('mcv.select.focus', this.
11                     xor(this.state.focusedIds, Number(feature.id)));
12             } else {
13                 this.state.controller.publish('mcv.select.focus', [
14                     Number(feature.id)]);
15             }
16         }
17     });
18 }

```

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

First we cache all layers in the internal state of the `Map` component. On each `mouseover` event, the `id` of the feature is published as `mcv.select.highlight` interaction. A `click` event is 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.

```

1 featureStyle(feature: geojson.Feature<geojson.GeometryObject>):
2     Leaflet.PathOptions{
3         const focused = (this.state.focusedIds.includes(Number(
4             feature.id)))
5         const fillColor = focused ? Color('red') : Color('blue');
6         let color = fillColor;
7         let weight = 2;
8         let dashArray = '3';

```

```

7  if (this.state.highlightedIds.includes(Number(feature.id))) {
8    weight = 4;
9    color = 'white';
10   dashArray = '';
11 }
12 return { color, weight, fillColor, dashArray };
13 }
```

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

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

```

1 pointToLayer(geoJsonPoint:any, latlng: Leaflet.LatLng){
2   return new Leaflet.CircleMarker(latlng);
3 }
```

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

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

5.4 MultiviewController

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

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

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

```

2
3 export class MultiviewController {
4   private _geojsonUrl: string;
5
6   public subscribe(msg:string, callback: (msg:string, data:any)
7     => void) {
8     PubSub.subscribe(msg, callback);
9   }
10
11  public publish(msg:string, data:any) {
12    if (msg === 'mcv.reconfigure.url'){
13      if((this._geojsonUrl == null) || (this._geojsonUrl !==
14        data)){
15        this._geojsonUrl = data;
16        fetch(data, {
17          credentials: "same-origin"
18        }).then((resp) => resp.json()).then((response) => {
19          PubSub.publish('mcv.reconfigure.geometry', response);
20          PubSub.publish(msg, data);
21        }).catch((err) => {
22          console.log(err);
23        })
24      } else {
25        PubSub.publish(msg, data);
26      }
27    }
28    public clearAllSubscriptions() {
29      PubSub.clearAllSubscriptions();
30    }
31 }

```

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

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

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

```
1  {
2      "type": "FeatureCollection",
3      "features": [
4          {
5              "type": "Feature",
6              "geometry": {
7                  "type": "MultiPolygon",
8                  "coordinates": []
9              },
10             "properties": {
11                 "NAME_1": "Baden-Württemberg",
12                 "state_code": "BW",
13                 "user_count_total": "34",
14                 "user_count_normalized": "0.10149253731343283"
15             },
16             "id": 0
17         },
18         {
19             "type": "Feature",
20             "geometry": {
21                 "type": "Polygon",
22                 "coordinates": []
23             },
24             "properties": {
25                 "NAME_1": "Bayern",
26                 "state_code": "BY",
27                 "user_count_total": "36",
28                 "user_count_normalized": "0.10746268656716418"
29             },
30             "id": 1
31         }
32     ]
33 }
```

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

We can use this data as input for our common VISUAL ANALYTICS PLATFORM, e.g. figure 5.4 shows the user distribution of RUNDFUNK MITBESTIMMEN.

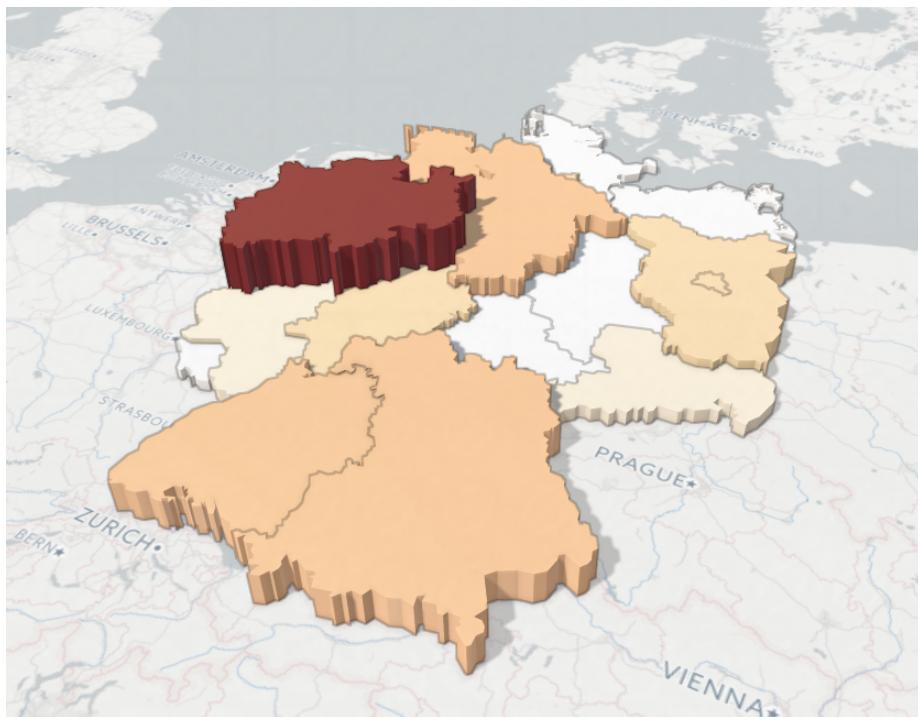


Fig. 5.4: User distribution of RUNDFUNK MITBESTIMMEN across German federal states

5.5 Implemented interactions

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

- *Select*: The user clicks on a building or region in a geographical map and all affected properties in the 2.5D tree map will be highlighted.
- *Explore*: The user clicks on a block in the 2.5D tree map and the viewpoint in the geographical map will be centered on relevant area.

- *Reconfigure*: The user selects a different feature set and the changes are reflected in both the geographical map (e.g. point instead of polygon geometries) and in the 2.5D tree map.
- *Filter*: The user double-clicks on a region in a geographical map and the 2.5D tree map will be based on data of only that region.

6

Evaluation

In this chapter the reference implementation is evaluated in three steps. 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 Evaluation of Use Cases

no section without text

6.1.1 Explain outliers with a geographical context

2.5D tree maps can have outliers, i.e. unusual local maxima of a certain attribute. If the attribute is mapped to e.g. height, a local maximum would be a block that protrudes an group of evenly heightened blocks. We can see such an example in Figure 6.1.

This example visualizes Berlin's gas stations. The 2.5D tree map on the left side has a layout based on the brand name of a gas station. I.e. a group of gas stations in the 2.5D tree map belong to the same brand like "Total" or "Aral".

The 2.5D tree map is perfectly suited to show a correlation based on the attributes itself. Apparently, there is a correlation of brand name and price. Gas stations of brand "Total" are rather expensive in general but with smaller price variations. "Aral" on the other hand is slightly cheaper than "Total" but some gas stations are very expensive. Gas stations of brand name "HEM", "Star" and "Sprint" are generally inexpensive.

However, the 2.5D tree map 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? If you have a look on the right side, you can see a possible explanation. Many of those outliers are gas stations located in the proximity of highways. These gas stations are generally a little more expensive and that applies especially to those stations of “ARAL”.

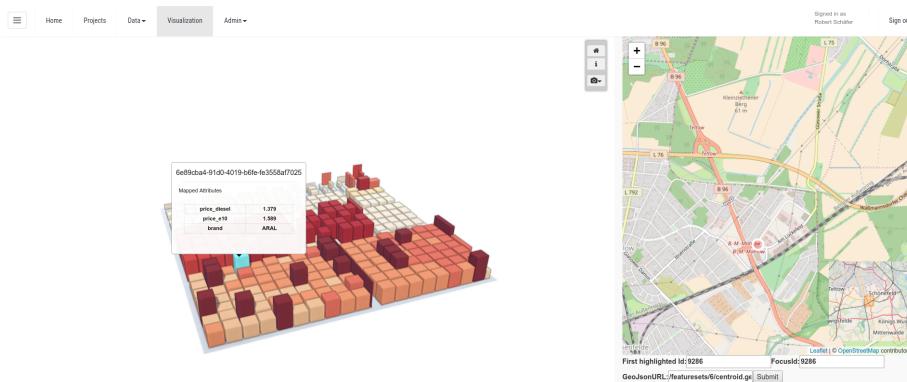


Fig. 6.1: The 2.5D tree map shows gas stations with a layout based on the brand name and height and colour mapped to price. Relatively expensive gas stations, with respect to the other gas stations of that brand, are often located next to highways.

6.1.2 Find an unusually cheap gas station in the center of Berlin

This use case demonstrates the benefit of a multiple select. Let's say, we're interested into unusually cheap gas stations which are not too far outside of Berlin. We would map attributes `price` and `distance` to colour and height, and would look for unusually coloured, protruding blocks in a region of the 2.5D tree map. If we press the control key and selectively click on each of those blocks, we collect a group of interesting gas stations. These gas stations are focused on in the geographical visualization as you can see in Figure 6.2.

6.1.3 Explain influx of sparsely populated administrative districts

This is another use case demonstrating the benefit of a multiple select plus the ability to connect interesting features with their geographical context. Figure 6.3 shows a 2.5D tree map with a common mapping: The layout is based on the population density, the large cluster at the top are sparsely populated districts.

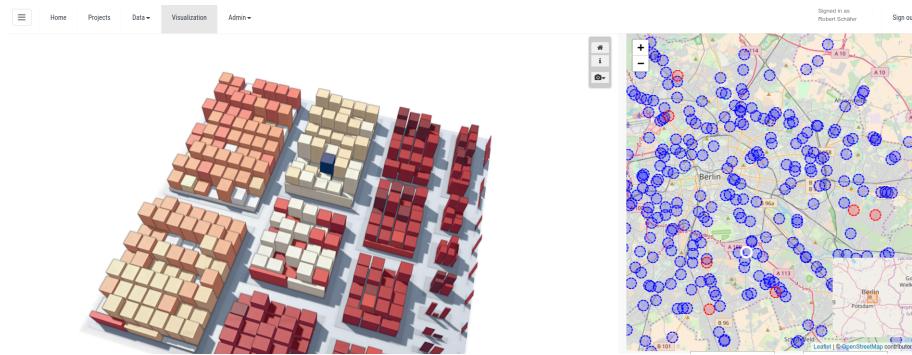


Fig. 6.2: A multi select allows to collect a group of interesting features.

The colour is based on the increase or decrease of inhabitants, a red colour indicating a decrease of inhabitants. The height of the blocks is mapped to unemployment rate.

The highlighted block and its two neighboring blocks catch our attention: As we can see on the right side, those three districts happen to be geographically related. They are all neighboring districts of Munich, the city with the highest rents in Germany.

We draw the following conclusion: Apparently people can not pay the rising rents in Munich anymore and start to move out in the rural areas that are somehow close to their work place in Munich. This conclusion coincided with the discovery of a geographical context that was not present in the mere data itself. A 2.5D tree map alone would not have been able to give us this insight.



Fig. 6.3: Three unsuspicious districts happen to be located next to Munich. Apparently people move out of Munich to avoid high rents.

6.1.4 Large districts by area with a high unemployment rate

Yet another use case to show the benefit of a multi select. In Figure 6.4 all rural districts with a high unemployment rate have been selected. Since these features are located in different groups in the 2.5D tree map that would not have been possible without the multi select. Figure 6.4 shows a correlation of unemployment in rural areas with the east of Germany.

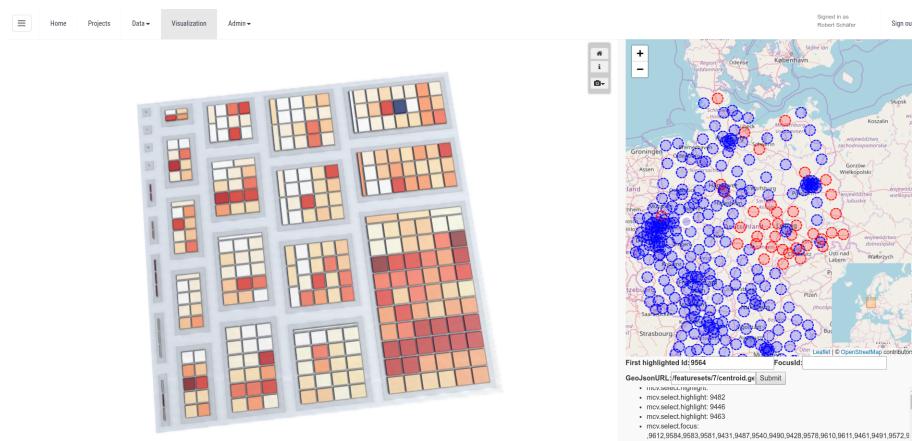


Fig. 6.4: The layout of the 2.5D tree map is based on area of the district, the colour based on the unemployment rate. We did a multi select of all rural districts with a high unemployment rate. We can see a strong correlation with a location in the east of Germany.

6.2 Performance Evaluation

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.

A couple of data sets were used in three different scenarios: (1) The 2.5D tree map without a geographical visualization, just publishing interactions, (2) an example application of the geographical visualization without a 2.5D tree map 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,

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

Table 6.1: Hardware specifications

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.

Name	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

Table 6.2: Data sets used for performance profiling

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 profiling of just the 2.5D tree map in Figure 6.5.

The Timeline identifies the event handler of the “mousemove” event to be the cause of this slow scripting. In this event handler, the currently highlighted point or polygon is picked in order to get the feature id. Obviously, this is a performance problem. It is worth noting, that not the coordination of the interaction is responsible of the performance problem, but an internal implementation of one visualization.



Fig. 6.5: During profiling of the “Immoscout” data set, only 16 frames per second get rendered

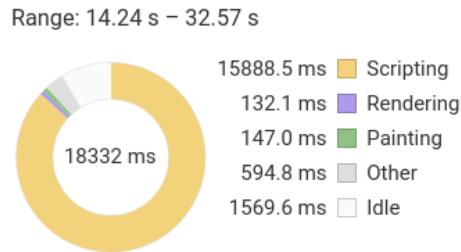


Fig. 6.6: Visualizing just the 2.5D tree map, almost the entire CPU time is spent during Scripting



Fig. 6.7: Every highlighting interaction comes with a spike of CPU time for rendering and painting. Focusing on a feature redraws the background and leads to an increase of painting time.

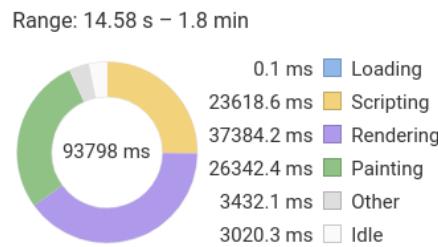


Fig. 6.8: The geographical visualization spends much more time during re-rendering and painting.

The profile summary looks totally different for the scenario of a 2.5D tree map along with a geographical visualization. Compared to just a 2.5D tree map 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.

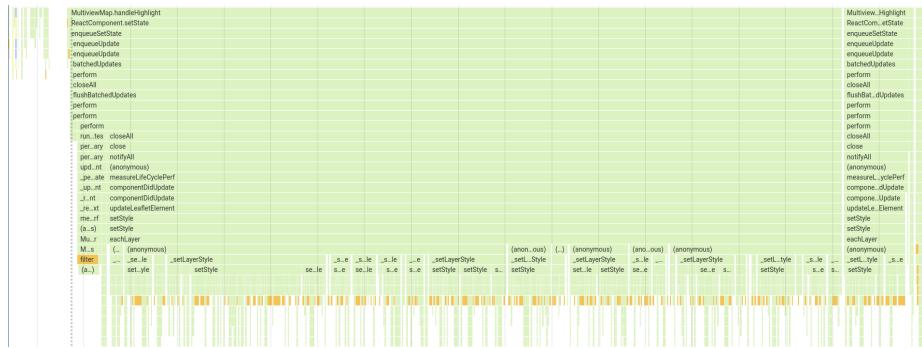


Fig. 6.9: Event handler `handleHighlight` takes about 90ms to finish. During this time, the list of features is iterated and `setStyle` executed.

Figure 6.9 shows that most of the time is spent in the event handler for a highlighted feature. LeafletJS iterates through all circle markers in order to update the style, i.e. change the stroke width. Therefore, this is a costly operation for a large number of features.

The data set “Landreise” is the second slowest data set. It is larger than “Immoscout” but has fewer features. As you can see in Figure 6.10 the geographical visualization can recover much faster if the data set has less features. About 10% of the CPU time is spent idling, see Figure 6.13.

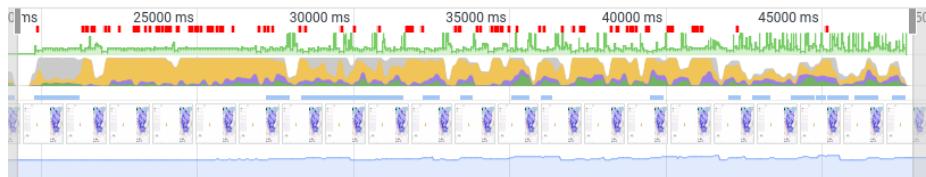


Fig. 6.10: The number of features is crucial for the time spent during a highlighting interaction.

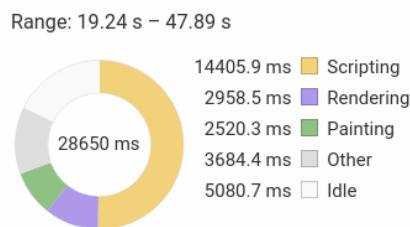


Fig. 6.11: Data set “Landkreise” shows less time spent on painting and rendering.



Fig. 6.12: A timeline without any significant drop of frames per second is achieved with the geographical visualization without 2.5D tree map of the “Landkreise” data set.

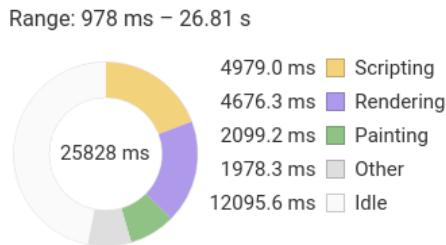


Fig. 6.13: Without a 2.5D tree map the CPU is idling for almost 50% of the time.

6.3 Evaluation of Requirements

In this section, the reference implementation is evaluated based on the requirements from Section 3.3.

Serialization of interactions depends on the published interaction data. The interaction type is just a string and therefore trivial to serialize. But the payload 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, i.e. 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 visualization 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 3.3 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 3.3 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.

Conclusion and Future Work

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.

Did we achieve our goals?

Is the concept sane in regarding the implementation?

List stuff which was not accomplished in this master thesis

References

- [1] Internet Engineering Task Force (IETF). *GeoJSON*. Aug. 1, 2016. URL: <http://geojson.org/> (visited on 11/20/2017).
- [2] World Wide Web Consortium (W3C). *Web components current status*. Feb. 13, 2017. URL: <https://www.w3.org/standards/techs/components> (visited on 07/24/2017).
- [3] Vladimir Agafonkin and OpenStreetMap contributors. *Leaflet*. Aug. 1, 2017. URL: <http://leafletjs.com/> (visited on 11/08/2017).
- [4] Benjamin Bach et al. “A Review of Temporal Data Visualizations Based on Space-Time Cube Operations”. In: *Eurographics Conference on Visualization (EuroVis 2014)*. 2014, pp. 23–41.
- [5] Jacques Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. Jan. 2010. ISBN: 978-1-58948-261-6.
- [6] Eric Bidelman. *google-map*. Nov. 18, 2017. URL: <https://www.webcomponents.org/element/GoogleWebComponents/google-map> (visited on 11/20/2017).
- [7] Staffan Bjork, Lars Erik Holmquist, and Johan Redstrom. “A framework for focus+ context visualization”. In: *Information Visualization, 1999.(Info Vis' 99) Proceedings. 1999 IEEE Symposium on*. IEEE. 1999, pp. 53–56.
- [8] Thomas Bladh, David A. Carr, and Jeremiah Scholl. “Extending Tree-Maps to Three Dimensions: A Comparative Study”. In: *Computer Human Interaction: 6th Asia Pacific Conference, APCHI 2004, Rotorua, New Zealand, June 29-July 2, 2004. Proceedings*. Ed. by Masood Masoodian, Steve Jones, and Bill Rogers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 50–59. ISBN: 978-3-540-27795-8. DOI: 10.1007/978-3-540-27795-8_6. URL: http://dx.doi.org/10.1007/978-3-540-27795-8_6.
- [9] Mike Bostock. *Crossfilter*. July 2017. URL: <http://square.github.io/crossfilter/> (visited on 07/10/2017).

- [10] Paul Le Cam. *react-leaflet*. Nov. 10, 2017. URL: <https://github.com/PaulLeCam/react-leaflet/commits/master> (visited on 11/17/2017).
- [11] MST Carpendale. “Considering visual variables as a basis for information visualisation”. In: (2003).
- [12] National Center for Chronic Disease Prevention and Health Promotion. *Choropleth Map of Obesity in the United States (1995 to 2008)*. 2010. URL: <https://homes.cs.washington.edu/~jheer//files/zoo/ex/maps/choropleth.html> (visited on 10/09/2017).
- [13] John Corbett. “Charles Joseph Minard, Mapping Napoleon’s March, 1861.” In: (2001).
- [14] Alan Dix and Geoffrey Ellis. “Starting Simple: Adding Value to Static Visualisation Through Simple Interaction”. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI ’98. L’Aquila, Italy: ACM, 1998, pp. 124–134. DOI: 10.1145/948496.948514. URL: <http://doi.acm.org/10.1145/948496.948514>.
- [15] Jürgen Döllner. *Visualization Techniques for Big Data*. July 2017. URL: <https://hpi.de/doellner/masterarbeiten/visual-software-analytics.html> (visited on 07/11/2017).
- [16] Patrick Th. Eugster et al. “The Many Faces of Publish/Subscribe”. In: *ACM Comput. Surv.* 35.2 (June 2003), pp. 114–131. ISSN: 0360-0300. DOI: 10.1145/857076.857078. URL: <http://doi.acm.org/10.1145/857076.857078>.
- [17] Stephen Few. “Data visualization for human perception”. In: *The Encyclopedia of Human-Computer Interaction, 2nd Ed.* (2013).
- [18] Pablo Figueroa, Mark Green, and Benjamin Watson. “A framework for 3D interaction techniques”. In: *CAD/Graphics*. Vol. 8. 2001, pp. 22–24.
- [19] Michael Friendly and Daniel J Denis. “Milestones in the history of thematic cartography, statistical graphics, and data visualization”. In: URL <http://www.datavis.ca/milestones> 32 (2001).
- [20] Simone Garlandini and Sara Fabrikant. “Evaluating the effectiveness and efficiency of visual variables for geographic information visualization”. In: *Spatial information theory* (2009), pp. 195–211.
- [21] Macro connection group. *The Observatory of Economic Complexity*. July 2017. URL: <http://atlas.media.mit.edu/en/profile/country/deu/> (visited on 07/03/2017).
- [22] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. “A tour through the visualization zoo”. In: *Queue* 8.5 (2010), p. 20.
- [23] Quan Ho. *Architecture and Applications of a Geovisual Analytics Framework*. May 2013, p. 78. ISBN: 978-91-7519-643-5.

- [24] Facebook Inc. *React, A javascript library for building user interfaces*. Sept. 7, 2017. URL: <https://facebook.github.io/react/> (visited on 09/23/2017).
- [25] Google Inc. *Polymer Project*. Sept. 22, 2017. URL: <https://www.polymer-project.org/> (visited on 09/23/2017).
- [26] Brian Johnson and Ben Shneiderman. “Tree-Maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures”. In: *Proceedings of the 2Nd Conference on Visualization '91. VIS '91*. San Diego, California: IEEE Computer Society Press, 1991, pp. 284–291. ISBN: 0-8186-2245-8. URL: <http://dl.acm.org/citation.cfm?id=949607.949654>.
- [27] Yehuda Katz. *handlebars*. June 22, 2017. URL: <http://handlebarsjs.com/> (visited on 09/23/2017).
- [28] Daniel A. Keim. “Information Visualization and Visual Data Mining”. In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (Jan. 2002), pp. 1–8. ISSN: 1077-2626. DOI: 10.1109/2945.981847. URL: <http://dx.doi.org/10.1109/2945.981847>.
- [29] Sam Kusinitz. *12 Reasons to Integrate Visual Content Into Your Marketing Campaigns*. July 18, 2014. URL: <https://blog.hubspot.com/marketing/visual-content-marketing-infographic> (visited on 07/08/2017).
- [30] Nada Lavrač et al. “Data mining and visualization for decision support and modeling of public health-care resources”. In: *Journal of Biomedical Informatics* 40.4 (2007). Public Health Informatics, pp. 438–447. ISSN: 1532-0464. DOI: <http://dx.doi.org/10.1016/j.jbi.2006.10.003>. URL: <http://www.sciencedirect.com/science/article/pii/S153204640600116X>.
- [31] Sam Li, Douglas Stockwell, and Andy Mutton. *What are web components?* Nov. 17, 2017. URL: <https://www.webcomponents.org/introduction> (visited on 11/20/2017).
- [32] Daniel Limberger et al. “Dynamic 2.5D Treemaps Using Declarative 3D on the Web”. In: *Proceedings of the 21st International Conference on Web3D Technology*. Web3D '16. Anaheim, California: ACM, 2016, pp. 33–36. ISBN: 978-1-4503-4428-9. DOI: 10.1145/2945292.2945313. URL: <http://doi.acm.org/10.1145/2945292.2945313>.
- [33] Richard E. Mayer. “Multimedia learning: Are we asking the right questions?” In: *Educational Psychologist* 32.1 (1997), pp. 1–19. DOI: 10.1207/s15326985ep3201_1. eprint: http://dx.doi.org/10.1207/s15326985ep3201_1. URL: http://dx.doi.org/10.1207/s15326985ep3201_1.

- [34] Andrew McAfee and Erik Brynjolfsson. *Big Data: The Management Revolution*. Oct. 2012. URL: <https://hbr.org/2012/10/big-data-the-management-revolution> (visited on 07/08/2017).
- [35] Thomas Nocke and Heidrun Schumann. “Meta Data for Visual Data Mining”. In: *Proceedings Computer Graphics and Imaging, CGIM’02*. Citeseer. 2002.
- [36] Thiago Poleto, Victor Diogho Heuer de Carvalho, and Ana Paula Cabral Seixas Costa. “The Roles of Big Data in the Decision-Support Process: An Empirical Investigation”. In: *Decision Support Systems V – Big Data Analytics for Decision Making: First International Conference, ICDSST 2015, Belgrade, Serbia, May 27-29, 2015, Proceedings*. Ed. by Boris Delibašić et al. Cham: Springer International Publishing, 2015, pp. 10–21. ISBN: 978-3-319-18533-0. DOI: 10.1007/978-3-319-18533-0_2. URL: http://dx.doi.org/10.1007/978-3-319-18533-0_2.
- [37] Severino Ribecca. *The Data Visualisation Catalogue*. Sept. 11, 2017. URL: <http://www.datavizcatalogue.com/index.html> (visited on 09/23/2017).
- [38] J. C. Roberts. “State of the Art: Coordinated Multiple Views in Exploratory Visualization”. In: *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007)*. July 2007, pp. 61–71. DOI: 10.1109/CMV.2007.20.
- [39] Morgan Roderick. *PubSubJS*. May 21, 2017. URL: <https://github.com/mroderick/PubSubJS> (visited on 11/09/2017).
- [40] Ben Shneiderman. “The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations”. In: *Proceedings of the 1996 IEEE Symposium on Visual Languages*. VL ’96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 336–. ISBN: 0-8186-7508-X. URL: <http://dl.acm.org/citation.cfm?id=832277.834354>.
- [41] Ben Shneiderman. “Tree Visualization with Tree-maps: 2-d Space-filling Approach”. In: *ACM Trans. Graph.* 11.1 (Jan. 1992), pp. 92–99. ISSN: 0730-0301. DOI: 10.1145/102377.115768. URL: <http://doi.acm.org/10.1145/102377.115768>.
- [42] Frank Treacy. *Getting Started with Ember and Data Down Actions Up*. 2017. URL: <https://emberigniter.com/getting-started-ember-cli-data-down-actions-up-tutorial/> (visited on 07/23/2017).
- [43] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. “Guidelines for Using Multiple Views in Information Visualization”. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI ’00. Palermo, Italy: ACM, 2000, pp. 110–119. ISBN: 1-58113-252-2.

- DOI: 10.1145/345513.345271. URL: <http://doi.acm.org/10.1145/345513.345271>.
- [44] Wikipedia. *Ember.js*. Sept. 20, 2017. URL: <https://en.wikipedia.org/wiki/Ember.js> (visited on 09/23/2017).
- [45] J. S. Yi, Y. a. Kang, and J. Stasko. “Toward a Deeper Understanding of the Role of Interaction in Information Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1224–1231. ISSN: 1077-2626. DOI: 10.1109/TVCG.2007.70515.