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
December 9, 2017

Abstract

2.5D tree maps are good visualizations for multi-dimensional and hierarchical data. If the data is also geographic, the geographic context is often lost when displayed in a 2.5D tree map. This is because the blocks of a 2.5D tree map are placed on the grid depending on the tiling algorithm. The geographic context is lost for sure if no geographic attribute is used as input for the algorithm. This loss of geographical context makes it difficult to understand and interact with the visualization. The blocks may be scattered across the 2.5D tree map so that the user can no longer recognize individual geographic areas or locations. In addition, blocks can no longer be easily grouped and selected based on their geographic proximity.

This thesis demonstrates if and how a geographic visualization combined with a 2.5D tree map can solve these problems. The focus is on interactions, how the individual visualizations communicate with each other, which information must be contained in messages and which data they relate to. These interactions are formalized and, based on this, a conceptual framework is developed that enables interactions for any data visualization. For this conceptual framework, a reference implementation is developed that couples a geographic visualization with a 2.5D tree map. The evaluation will be based on typical application scenarios, formal criteria and requirements as well as a performance analysis.

Zusammenfassung

2.5D tree maps sind gut geeignete Visualisierungen für multi-dimensionale und hierarchische Daten. Handelt es sich bei den Daten auch um geographische Daten, geht der geographische Zusammenhang bei der Darstellung in einer 2.5D tree map oft verloren. Das liegt daran, dass die Blöcke einer 2.5D tree map in Abhängigkeit vom Tiling-Algorithmus auf dem Raster platziert werden. In jedem Fall geht der geographische Kontext verloren, wenn kein geographisches Attribut als Eingabe für den Algorithmus verwendet wird. Dieser Verlust des geographischen Kontextes erschwert die Verständlichkeit und die Interaktivität der Visualisierung. Die einzelnen Blöcke werden unter Umständen verstreut dargestellt, sodass der Nutzer nicht mehr nachvollziehen kann, um welche geographischen Gebiete oder Orte es sich handelt. Außerdem können Blöcke nicht mehr leicht anhand ihrer geographischen Nähe gruppiert und selektiert werden.

Diese Arbeit zeigt, ob und wie eine geographische Visualisierung kombiniert mit einer 2.5D tree map diese Probleme lösen kann. Der Fokus liegt dabei auf Interaktionen, wie die einzelnen Visualisierungen miteinander kommunizieren, welche Informatiknen in Nachrichten beinhaltet sein müssen und auf welche Daten sich diese beziehen. Diese Interaktionen werden formalisiert und darauf aufbauend ein konzeptuelles Framework entwickelt, welches Interaktionen über beliebige Datenvisualisierungen ermöglicht. Für dieses konzeptuelles Framework wird eine Referenzimplementierung entwickelt, welches eine geographische Visualisierung mit einer 2.5D tree map koppelt. Die Evaluierung wird anhand typischer Anwendungsszenarios, formaler Kriterien und Anforderungen sowie einer Leistungsanalyse durchgeführt.

Contents

1

Introduction

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

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

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

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

1.1 Motivation

Many research papers in the field of coordinated multiple views have focused more on visual representations than interaction aspects. Even though interaction aspects give a great user experience and enable the user to find complex connections involving multiple views. Ho [22] assumes this may “originate from

the fact that the implementation of interaction techniques and interactive features normally takes much more time than the implementation of visual representations". Obviously, there is a lot of existing research in the area but a lack of research regarding interactions in particular. This could also explain why hardly any general-purpose implementation exists to coordinate multiple views, that focuses on interactions. It is for this reason that this thesis develops an interaction model for coordinated multiple views, so future implementations can use this model as a specification.

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

1.2 Problem Statement

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

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

1.3 Hypothesis

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

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

1.4 Contributions

The contributions of this work are the following:

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

1.5 Structure of the Work

In Section 2 we introduce basic terminology of coordinated multiple views and the theoretical background in this area of research. This Section also covers the

state of the art and research on multiple views and coordinated interactions. In Section 3 we analyze a set of data visualizations and their interactions by example. Each interaction is further examined to identify the relevant information that need to be communicated in a coordinated multiple view. The gained knowledge from that section is used in the following Section 4 to develop a conceptual framework that can be used for future implementations of coordinated multiple views. This conceptual framework includes a data model shared between all views, a suggested communication protocol as well as a formalization of an interaction. In Section 5 we describe the implementation of the conceptual framework for the present use case. This implementation serves as reference implementation, to prove feasibility of the conceptual framework. The implementation includes the necessary interactions to demonstrate the advantage of coordinated multiple views for hierarchic and geographic data. It also serves to validate or invalidate the hypothesis in Section 1.3. In Section 6 the implementation is tested for typical visual analytics tasks and it is demonstrated that additional value can be generated from the combination of 2.5D tree map and geographic visualization. A performance analysis of the implementation is carried out to demonstrate the feasibility of the conceptual framework.

Check design criteria or rephrase/delete this sentence

Based on the test data sets, the coordinated multiple view implementation is examined and evaluated for design criteria [43] and general usability aspects [37]. The types of evaluation are therefore:

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

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

2

Related Work and Foundations

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

2.1 Information visualization

Information visualization is a means of visual communication and have steadily developed since the 16th century [19]. It is a generic term, expressing all kinds of effort to put data into visual context to help people understand the significance of data [39]. Information visualization today goes beyond standard charts and graphs used in spreadsheet applications and covers also infographics, heat maps, geographic visualizations and tree maps. Otherwise abstract information is visually represented, making complex data more accessible, understandable and usable.

Kusinitz [27] mentions that the human brain processes visual information 60,000 times faster than text and visual content makes up even 93% of all human communication. According to the Interaction Design Foundation the purpose of data visualizations is twofold: Sense-making and communication.

Statistical information is abstract and in data visualization “we must find a way to give form to that which has none.” [17] Successful data visualizations helps the human user to derive knowledge and meta data from the visualization itself. Nocke and Schumann [34] call this “visual data mining”.

2.2 Data-driven decision support systems

Data-driven decision support systems are applications to support businesses and organizational decision-making activities in which data visualizations play a key part [28] [35]. A common expression by impatient managers who can not afford to wade through lengthy reports has even become the title of a book about decision support systems: Stephen Few's "Show me the numbers".

In the business context, sales managers demand a quick access on the latest data with all relevant visualizations at once. Examples of this kind of decision support systems are called e.g. "decision cockpit" or "business sphere"[12].

We can expect to see these technologies more in more in business applications. McAfee and Brynjolfsson [33] from the MIT Center of Digital Business showed that organizations driven most by data-based decision making had 4% higher productivity rates and 6% higher profits.

However, little research has been done regarding the performance of coordinated multiple views in the field of decision making. There might be a great potential. In 1997 Mayer [32] conducted eight studies to compare the effect of using multimedia on university students. The studies showed that when using combined visual and verbal explanations the generation of creative problem solutions increased by an average of more than 50%.

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

2.3 Geovisualization

The umbrella term "Geovisualization" covers visualization techniques for the "visual exploration, analysis, synthesis and presentation of geospatial data (any data having geospatial referencing)" [31]. The entities of geospatial data may include buildings, streets, landmasses, terrain, administrative districts and also moving entities like cars.

2.3.1 Visual Variables

French cartographer Jaques Bertin introduced seven visual variables in 1967 [4]. 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 [10] explains in detail their use in computational information instead of

printed cartography. Garlandini and Fabrikant [20] put these visual variables under systematic validation procedures. The authors conclude that the variable `size` provides the most accurate and efficient performance while the variable `orientation` provides the least performance. Bertin's visual variable play a role

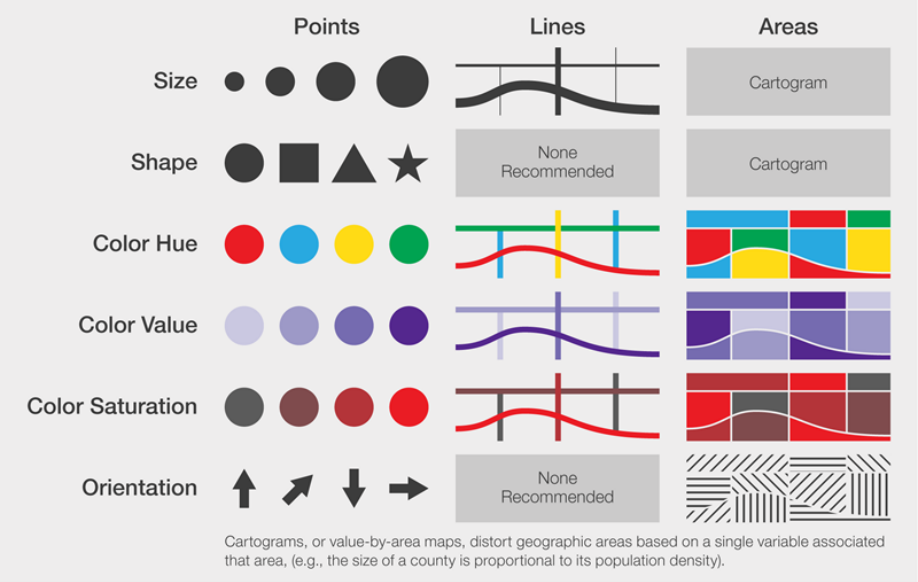
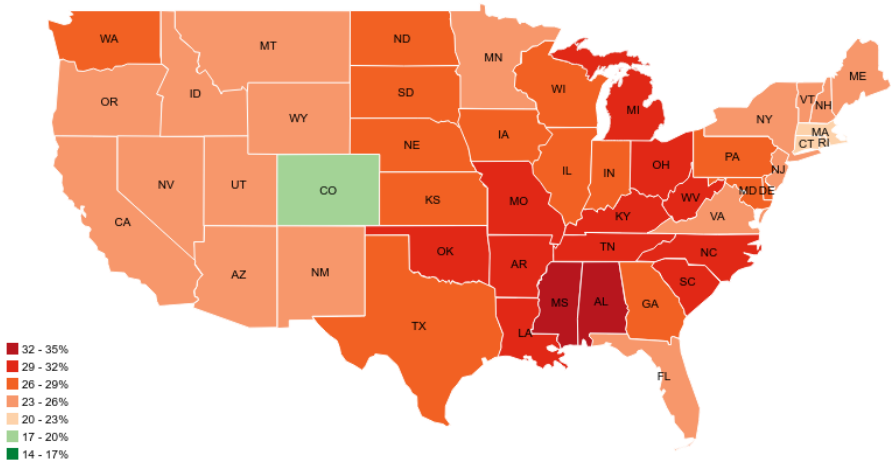


Fig. 2.1: Bertin's [4] original visual variables.

in interaction aspects of coordinated multiple views as they are used to communicate the effect of an interaction. A highlighted data point can be highlighted by changing the colour of an area or increasing the size of a point.

2.3.2 Choropleth maps

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



National Center for Chronic Disease Prevention and Health Promotion [11]

Fig. 2.2: Choropleth Map of Obesity in the United States in 2008. Present of population classified as "obese" (Body Mass Index greater than 30), by state.

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

2.4 Information Visualization of Hierarchic Data

The visualization of hierarchical data has a long tradition. The traditional visual representation of a tree is a directed graph with the root node at the top, as seen in Figure 2.3. An common use case is a directory tree of a file system,

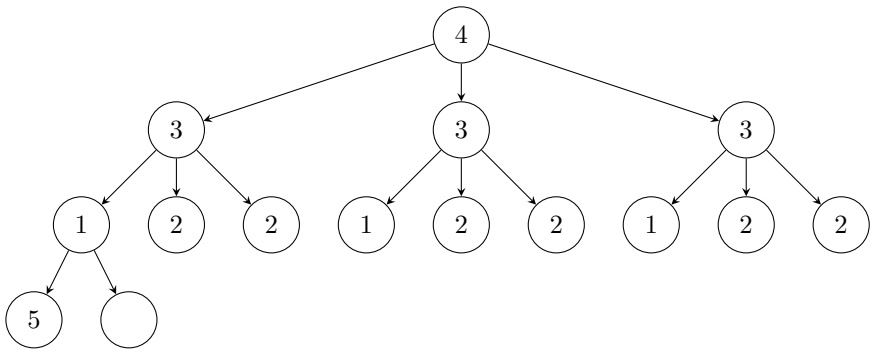


Fig. 2.3: Traditional visualization of a tree in form of a directed graph with edges and nodes and the root node at the top

e.g. a file browser or the command line utility `tree` on UNIX based operating systems. As Shneiderman [41] mentions, this visualization becomes increasingly large when displaying more than one level and soon exceeds the entire screen size.

2.4.1 Tree Maps

Johnson and Shneiderman [24] propose the tree map visualization technique, in which each node is a rectangle whose area is proportional to a specified dimension. In tree maps every node is visualized as a tile. The membership relationship is expressed with tiles containing other tiles, thus representing the hierarchy.

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

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

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

If tree maps are used to visualize geographic data like municipalities, real estates or streets, the placement of nodes depends on the tiling algorithm and not the geographic location on a map. Let’s say a tree map visualizes a hierarchy of federal states and municipalities with the area of each tile mapped to the total population of administrative district. Even if the membership hierarchy is matched by the tree map, the placement of districts in the same hierarchy level is based on their total population and not their geographic location.

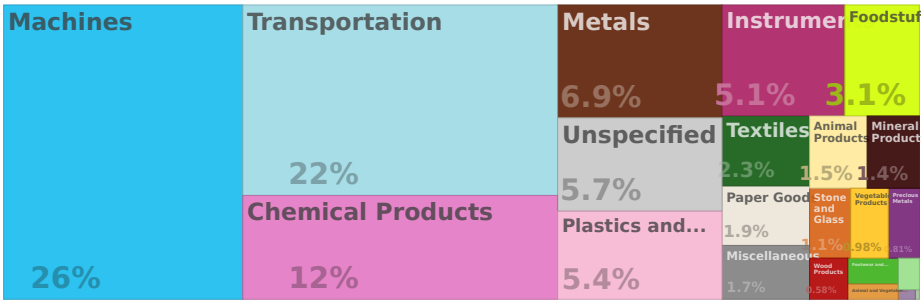


Fig. 2.4: A two dimensional tree map of Germany's foreign trade quota of exports, showing only the first hierarchy level.

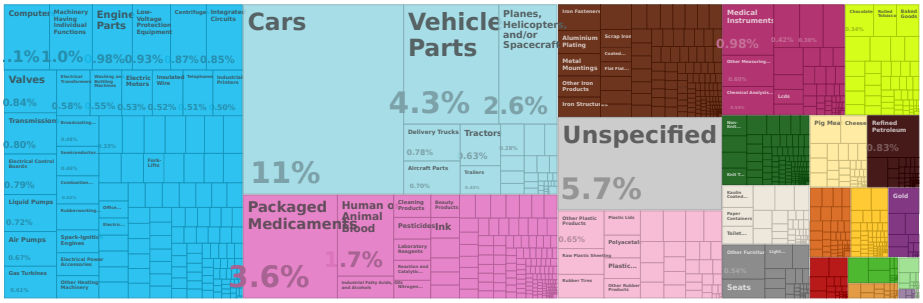


Fig. 2.5: A two dimensional tree map of Germany's foreign trade quota of exports, showing a hierarchy of depth "HS4"

2.4.2 3D tree maps and 2.5D tree maps

3D tree maps are a concept introduced by Bladh, Carr, and Scholl [7] in 2004. The authors transfer the concept of tree maps from two dimensional into three dimensional space, transforming tiles to blocks. They introduce "StepTree" [7], which is a three dimensional tree map to display a directory layout of a file system. It "differs from tree maps 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. Blocks can superimpose each other, forcing the user to navigate the view point. The navigation of the view point itself is an increase of complexity not present in two dimensional tree maps.

The term 2.5D tree map was coined by Limberger et al. [30] in 2016. A 2.5D tree map is just an ordinary 3D tree map, but it has all blocks attached to the ground, or more specifically, attached to the parent block. We can see an example of a 2.5D tree map in figure 2.6. All three dimensional tree maps in this thesis are in fact 2.5D tree maps and therefore this term is used for the rest of this work.

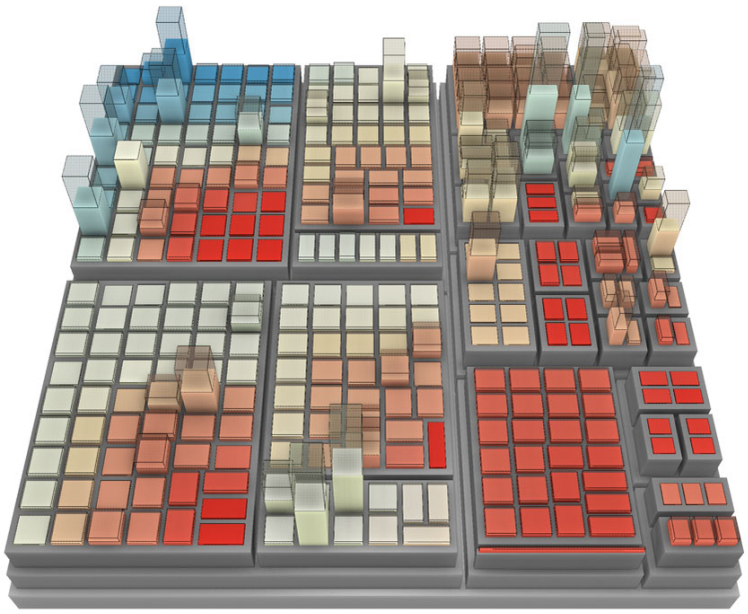
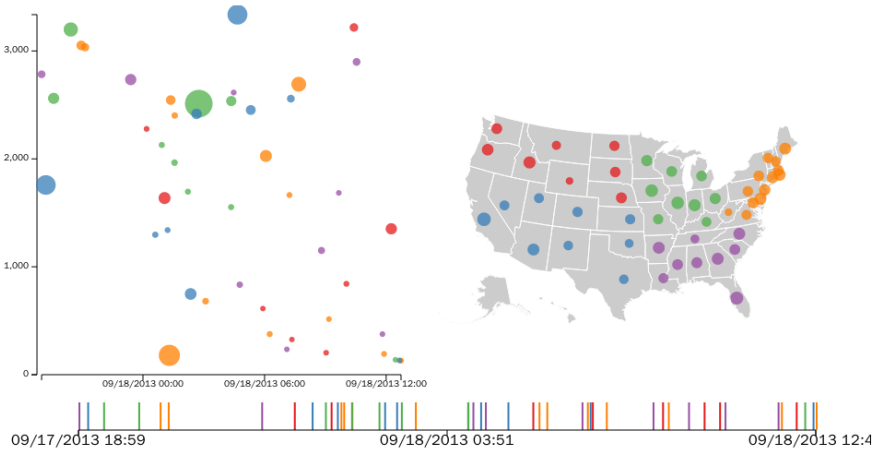


Fig. 2.6: Example of a 2.5D tree map

2.5 Coordinated Multiple Views

Coordinated multiple views are a combination of data visualizations of the same data set in multiple views, often side-by-side. According to Roberts [37] coordinated multiple views is just “a specific exploratory visualization technique that enables users to explore their data”. Because “the overall premise for the technique is that users understand their data better if they interact with the presented information and view it through different representations.” [37]

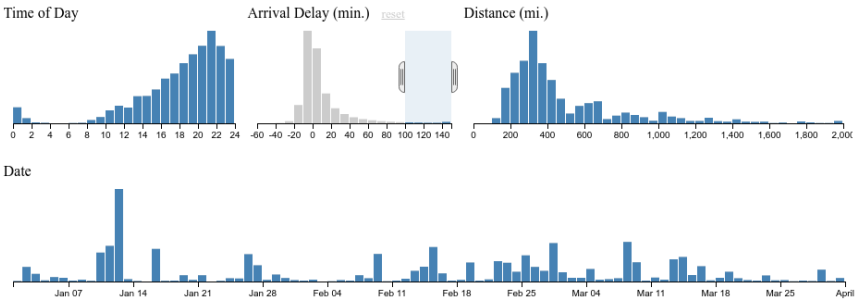
We can see an example of a coordinated multiple view in Figure 2.7 It displays spatial and temporal attributes of pictures from a picture database, as well as continuous attributes like popularity and number of comments. The user can



Dukevis[15]

Fig. 2.7: coordinated multiple view displaying the popularity, number of comments, location and time of pictures in a picture data base

move the mouse cursor over each item in the scatter plot and the graduated symbol map and the corresponding item is highlighted with a larger stroke in all other views. On the time line below, the user can also filter for pictures in a certain time frame by dragging the mouse from lower to upper limit.



Crossfilter[8]

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

2.5.1 Brushing and Linking

Brushing and linking is a common interaction pattern found in coordinated multiple views and often a crucial part of these visualizations. “The technique of brushing is the principle approach, where elements are selected (and highlighted) in one display, concurrently the same information in any other linked display is also highlighted.”[37]

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

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

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

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

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

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

2.6 Technologies

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

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

Web components is a recent standard of the W3C[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 [29]. If coordinated multiple views are implemented in JavaScript based web applications, web components are a promising choice, to allow arbitrary views to be put together.

GlimmerJS is the rendering enginge of EmberJS[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[25], 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 [23]. With 18,469 stars on Github it is the most popular framework for web components at the time of writing. Polymer has a large community and comprehensive documentation and therefore more suitable than GlimmerJS to build coordinated multiple views.

ReactJS is an open-source JavaScript library to build user interfaces and allows to create reusable UI components. React renders HTML on the client, it changes the page without reloading the page. The framework corresponds with the View in the Model-View-Controller pattern. React components are structured hierarchically, with each component having dedicated responsibility. React is explicitly not implementing web components and is not going to implement web components in the future. It has, in return, a well-known way of integrating the component framework into a legacy application built with e.g. jQuery. Along with its major advantage of easy integration, it has a striving community, lots of documentation and tutorials and it is well tested.

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

2.7 Interaction Theory

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

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

2.8 Interaction Categories

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

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

In 2002, Keim [26] comes up with the following classification: (1) Dynamic *projection* to show all combination of data attributes mapped to the axis of a diagram, (2) focus on a smaller subsets by *filtering* out parts of the data, (3) *zoom* into a subset of the data and get a higher level of detail, (4) drill-down operations to preserve an overview of the data are called *distortion* (5) and finally *link and brush* visualizations, to highlight the same data points in multiple visualizations.

The most recent classification was done in 2007 by Yi, Kang, and Stasko [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* to show more or

less detail, (6) *filter* to show something conditionally, (7) *connect* to show related items.

It is noticeable that all of these classifications of interactions are redundant. In this work the classification of Yi, Kang, and Stasko [45] is used for the remaining parts because it is the most recent classification and it is based on the precursors.

Shall I give an example for each category of Yi, Kang, and Stasko [45] classification?

2.9 Formalization of Interactions

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

ITlib[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, p. 2]. These filters are the smallest process unit in the data flow. Composed of input and output ports, they communicate with other filters, to receive data input from predecessors and send data output to successors.

The framework specifies and stores the interaction techniques along with its filters, the execution model and the scene in XML documents. The authors chose XML because it can be parsed easily and they generate code in order to target various virtual reality toolkits and environments.

Even though the system describes interactions in an abstract way, the domain of the framework is clearly the interaction of a human body within a 3D virtual reality. Certain assumptions are made, including the data model, which is the 3D scene, and human computer interaction devices, like the user's hand or the user's head.

The goal is not to better understand the data, which is the common goal in data visualizations. In this case, the data model is the 3D scene and the goal is to manipulate the 3D scene.

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

A framework for Focus+Context Visualization by Bjork, Holmquist, and Redstrom [6] is one of the few formalizations of interactions in data visualizations.

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

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

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

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

If I affects $[D]$ we can manipulate the underlying data set. Examples would be changes in a spreadsheet editor, or a change of the start and end date of an appointment in a calendar.

If V is affected by I the user can manipulate IV in order to change the way $[D]$ is represented. This statement holds e.g. for an interaction in which the user increases the visible level of hierarchy in a tree map, which is an `abstract/elaborate` interaction according to Yi, Kang, and Stasko [45]. The effect of such an interaction is depicted by the change from Figure 2.4 to Figure 2.5 in Section 2.5.

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

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

Focus+context visualizations are second-level visualizations. An example given by the authors is the "rubbersheet" visualization, that visually distorts a first-level visualization similar to a magnifier. Regions of primary interest are distorted to appear magnified, while the remaining regions are minified.

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

3

Analysis

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

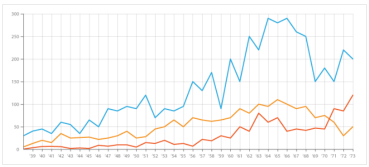
Finally in Section 3.5 the advantages and disadvantages of several web frontend frameworks are contrasted. A final decision is made which framework will be used for the implementation of the conceptual framework.

3.1 Single View Interactions

The data visualization catalogue by Severino Ribecca lists many of the most used data visualizations[36]. This section covers a list of common data visualizations from that catalogue as examples. The expected data structure and a list of possible interactions are systematically analysed.

3.1.1 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. Line graphs visualize one or many series of data in parallel and therefore the expected data format is *tabular*.

Line graphs are drawn in a Cartesian coordinate system, connecting subsequent points to each other. Thus, (1) position (2) orientation and (3) texture are constrained by the nature of the visualization. However, an interaction with the line graph can alter the (1) shape (2) color or (3) size of lines to communicate an interaction. It is further possible to highlight either the entire series of data or a single data point within that series, e.g. changing the shape and size of the point. Table 3.1 shows a list of plausible interactions in a line graph.

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

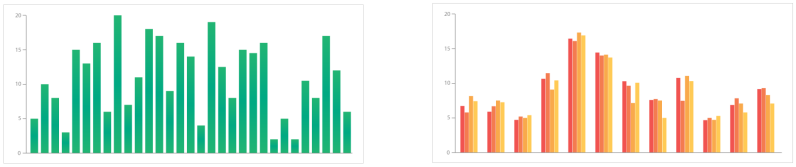
Table 3.1: Plausible interactions for line graphs

3.1.2 Bar Charts and Multi-Set Bar Charts

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.



(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

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 lists some possible interactions.

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

Table 3.2: Plausible interactions for bar charts

3.1.3 Histograms

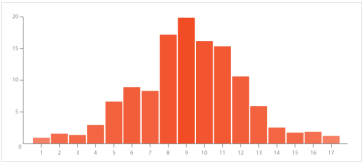


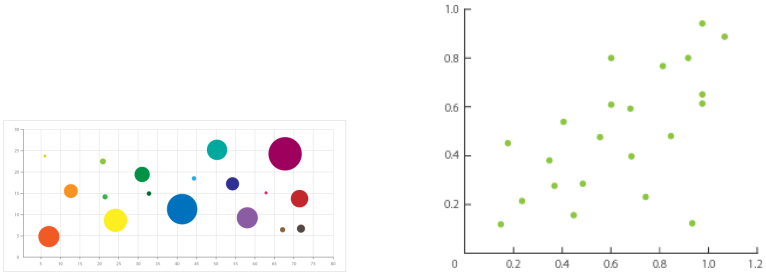
Fig. 3.3: A histogram is a bar chart over a continuous interval

Histograms visualize the distribution of data over a continuous interval or a certain time period. A special type is the population pyramid, which is a pair of back-to-back histograms, one for each sex, as seen in Figure 3.6.

Histograms and bar charts expect the same kind of data, i.e. a *tabular* data structure. Almost the same interactions as in Table 3.2 can be applied to histograms,

except a re-ordering of bars along the x-axis. This is not possible, because the histogram constrains the position of bars along the interval.

3.1.4 Bubble Charts and Scatter Plots



(a) Bubble Chart (b) Scatter plot

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

Both bubble charts and scatter plots are techniques to visualize continuous values from two data attributes. Points are placed with the two attributes in Cartesian coordinates in order to detect relationships and correlations.

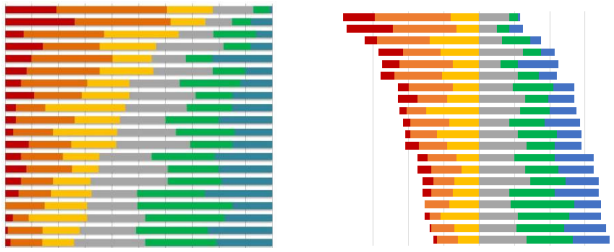
In case of bubble charts, each point is displayed as a bubble with a third value encoded in the size of bubbles. It is even possible to encode a fourth value in the colour of the bubble.

Like line graphs, bar charts and histograms, a scatter plot expects *tabular* data. Each data point can take up to four values (in case of a coloured bubble chart). As we can see in Table 3.3, interactions also include a zooming and movement of the viewpoint.

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

Table 3.3: Plausible interactions for bubble charts

3.1.5 Stacked Bar Charts



(a) Stacked bar chart

(b) Stacked bar chart with baseline

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

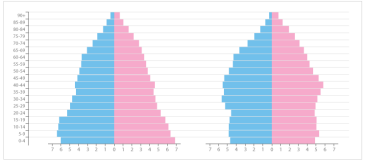


Fig. 3.6: A population pyramid is a pair of histograms and can also be modeled as a stacked bar chart

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

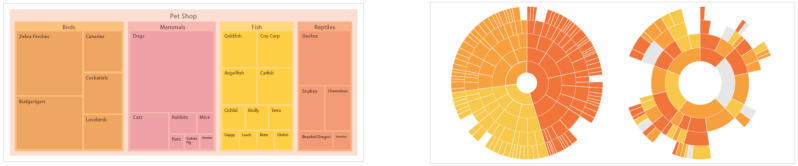
A stacked bar chart also expects *tabular* data.

If the stacked bar chart has a baseline, often the algebraic sign of the numeric value defines the placement of the segment on the left or on the right side. Table 3.4 shows possible interactions, including the highlighting of a data point, a change of color mapping or a reordering of the baseline.

Category	Description	Required information
Select	Highlight a bar	id of data point
Encode	Change mapping of category to colour	id of data series + colour
Reconfigure	Sort by attribute	name of 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

Table 3.4: Plausible interactions for stacked bar charts

3.1.6 Hierarchical visualizations



(a) Tree map (b) Sunburst diagram

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

Tree maps are great to show hierarchical data without ever exceeding the available screen. Each data point is represented as a tile. Unlike a tree map a hierar-

chical ring diagram or sunburst diagram shows each level of the underlying tree as a series of rings.

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

As these visualization techniques are about hierarchies, the visible, maximal depth of the tree may be increased or decreased. Again, interactions could include a highlighting of data points and a change of color encoding. Both visualizations may show only a subtree. E.g. a click on a box in the tree map opens another tree map focused on the subtree. Similarly, a click on a slice of the ring would surround the most external ring with the child nodes of the parent node. Table 3.5 gives a more comprehensive list of interactions.

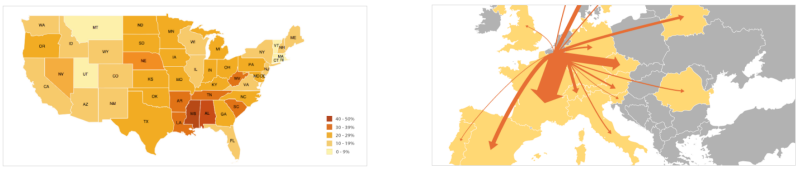
Category	**Description**	**Required information**
Select	Highlight a node	id of data point
Explore	Use another node as root of the visible tree	id of data point
Encode	Change mapping of category to colour	id of data series + colour
Reconfigure	Change data attribute used for layout	name of data attribute
Reconfigure	Sort by attribute	name of data attribute
Reconfigure	Specify order	ordered list of ids of data points
Abstract/Elaborate	Specify maximum depth of visible tree	number of hierarchy levels

Table 3.5: Plausible interactions for hierarchical visualizations

3.1.7 Geographic Data Visualizations

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

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



(a) Choropleth map (b) Flow map

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

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

Table 3.6: Plausible interactions for geographic visualizations

Activity diagrams

(a) Calendar (b) Gantt chart

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

In activity diagrams, each feature is represented as a rectangle, with the duration of the activity mapped to size and position. Calendars and gantt charts could not only read the data from the data source, but also add new features to the data set or update metadata of a feature, e.g. the progress of the activity. Calendars and gantt charts expect *tabular* data, although data points might reoccur on a regular schedule. So some data points, i.e. events, might repeat infinitely.

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

Table 3.7: Interactions for temporal visualizations

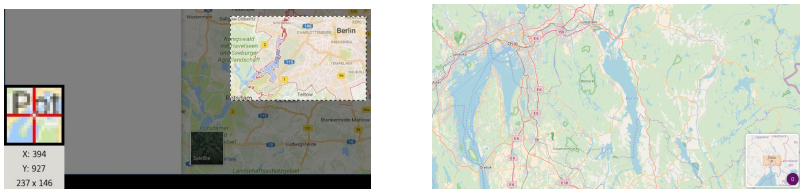
3.2 Multiple View Interactions

This section covers examples of multiple data visualizations. Combination of views are often very specific to certain use cases. That's why the examples in this section are tied to a use case in contrast to the more generic examples in Section 3.1.

3.2.1 Detail view and Mini View

Detail views show a magnification of the surrounding area of a cursor in a secondary view. The resolution of the primary view is too high for the user to distinguish individual pixels. The detail view helps to select items. Popular use cases are screenshot tools, as seen on the left in Figure 3.10, or image processing application.

The opposite of a detail view is the mini view. You can see an example on the right side of Figure 3.10. This secondary view shows a larger surrounding of the primary view, if the primary view shows only a section of the entire space. Use cases are geographic visualizations and therefore the expected data structure is a map with coordinates of the viewpoint and a zoom factor for each view.



(a) Detail view (b) Mini view

Fig. 3.10: The screenshot-tool “Shutter” shows a magnified detail view of the area around the mouse cursor in the lower left corner of the screen. The opposite approach is e.g. “Leaflet-Minimap” that shows the map at a larger scale in the lower right.

Category	Description	Required information
Explore	Move secondary viewpoint	Latitude + Longitude
Explore	Move primary viewpoint through secondary view	Latitude + Longitude
Explore	Accelerate scrolling via secondary view	Zoom factor

Table 3.8: Plausible interactions for detail and mini views

3.2.2 Timeline

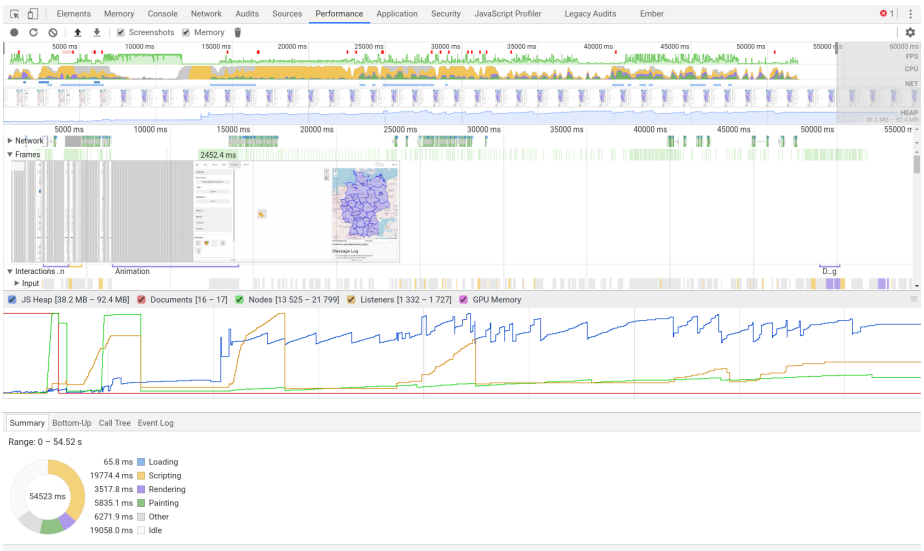


Fig. 3.11: Multiple views of Chromium’s runtime analysis tool can be focused on a selected time span.

Figure 3.11 shows the runtime analysis of the Chromium browser. Multiple views show different granularities of the same data. At the top, the entire profiling time frame is visualized, with a screenshot of one step in the middle. The development of memory is displayed on the third row while a summary is on the bottom. The user can select a window on the top line and the data for the other views is reduced according to the selected window. A multiple visualization like that expects *temporal* data.

Category	Description	Required information
Filter	Filter data by time span	Upper and lower limit
Select	Display screenshot a certain time	Timestamp

Table 3.9: Plausible interactions for timelines

3.3 Requirements

In this subsection, a list of formal requirements is imposed on a conceptual coordinated multiple view framework. These requirements can be checked in Section 6 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 to undo the effect of an interaction if possible. 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.

Extensibility qualify how much time and effort is needed in order to develop new components for the coordinated multiple view framework.

Write about software extensibility not data extensibility

Maintainability means in our case, how much other views are impacted by a change of an interaction in one view and how error-prone the framework is. In general it is hard to measure maintainability. For the coordinated multiple

view framework we want to measure the (1) lines of code and the (2) cycliomatic complexity. We will try to find a means to measure (3) cohesion and (4) coupling in the framework. The assumption is that the amount of shared data between views is an indicator for high coupling.

3.4 Existing Interactions

In this section, the existing interactions are classified according to the classification by Yi, Kang, and Stasko [45] of Section 2.8.

In the existing VISUAL ANALYTICS PLATFORM possible interactions can be categorized into the classes *select*, *explore*, *reconfigure*, *encode* and *filter*.

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

The user can *explore* the map in the usual manner: If the user drags with the mouse on the map, a panning operation is performed with the viewpoint focused on Germany, i.e. the camera moves around like a turntable. The zoom factor can be changed by scrolling with the mouse on the canvas of the map.

Encode and *reconfigure* techniques are performed through 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 to a range of continuous values. When the user drags the slider, the items in the map on the right side are *filtered* accordingly.

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. React does not support web components.

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.

Listing 3.1: An example of string based attributes of web components [5]

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

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

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

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

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

Figure 3.12 shows the pros and cons of each framework for the use case.

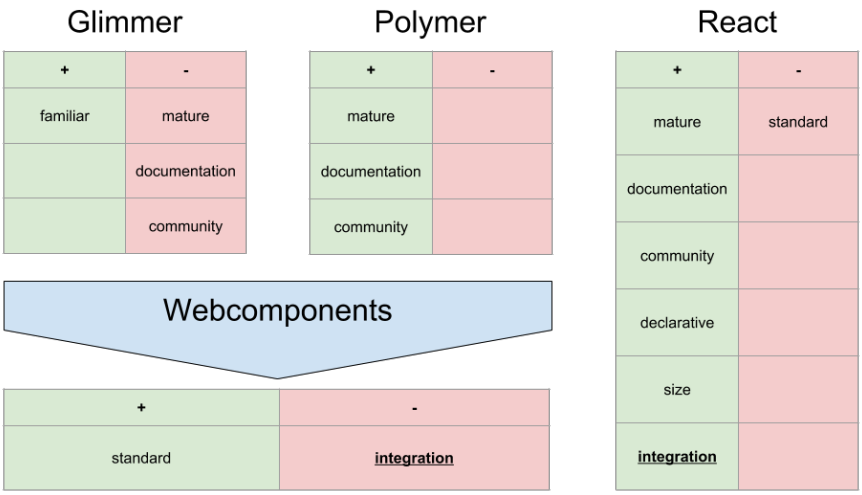


Fig. 3.12: Comparison of component based web frameworks, advantages highlighted in green, disadvantages highlighted in red.

4

Conceptual Framework

Based on the findings in Chapter 3 this chapter specifies a formal language and a conceptual framework for coordinated multiple views. The 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. Each interaction from Chapter 3 is formalized as a mathematical function and these functions are grouped if they have the same input and output. As a last step, the communication pattern of the formalized language is layed out.

4.1 Formalization of an interaction

The formalization of an interaction I is defined as follows:

$$I(T, M(C, P, IS), E) \tag{4.1}$$

For a trigger T, a message M and the effect E. The message M is composed of a category C, an application specific purpose P and an interaction subject IS. The part of the interaction which is shared among multiple views is the message M.

The category C 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.

4.1.1 Trigger

A 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.

4.1.2 Effect

The effect is the change of the visual representation subsequent to the interaction. In order to be perceivable by the user, the interaction must have some visual effect, 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*. Obviously visual effects are not shared, as it depends on the visualization technique if visual variables are available to express a visual effect. A re-order interaction will not have an effect in bubble charst, as position is constrained by the type of visualization.

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

$$I(T, E) \tag{4.2}$$

The meaning of the interaction in a single 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*.

This implicit meaning gets explicit in coordinated multiple views so that other views are able to react appropriately.

4.1.3 Subject

A subject refers to the target of the interaction. We must define what data or meta-data is affected by the interaction. E.g. when a user moves the mouse

cursor on a line in the line diagram, that could highlight the *data point* under the cursor as well as the entire *data series*. Therefore we call the object affected of an interaction the interaction *subject*. A subject can be a data point, a list of data points, a position of the viewpoint, a certain order of attributes or a mapping of attributes to visual variables.

4.1.4 Purpose

The purpose refers to the application specific 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 *purpose*. The purpose describes the interaction in the context of a task or an intention that is intrinsic to the application the interaction happens in.

4.1.5 Category

An category is the declaration and definition how the subject of the interaction should be changed. The aforementioned explicit meaning of the interaction is the smallest unit of information of the interaction.

Some examples of categories include: Selection, Deletion, Point-of-Interest, Filtering, Reordering, Re-encoding. Categories can be classified with the interaction categories of Yi, Kang, and Stasko [45].

4.1.6 Message

A message is the required set of information to coordinate an interaction across multiple views.

4.2 Shared data model

To account for the various data structures, we use an abstract data model that is inclusive 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.

4.3 Predefined categories

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

These function are derived from specific interactions of the examples in Section 3. Domain and range of these functions refer to the objects defined in the data model in Section 4.2. To declare the functions explicitly, we define a couple of sets:

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

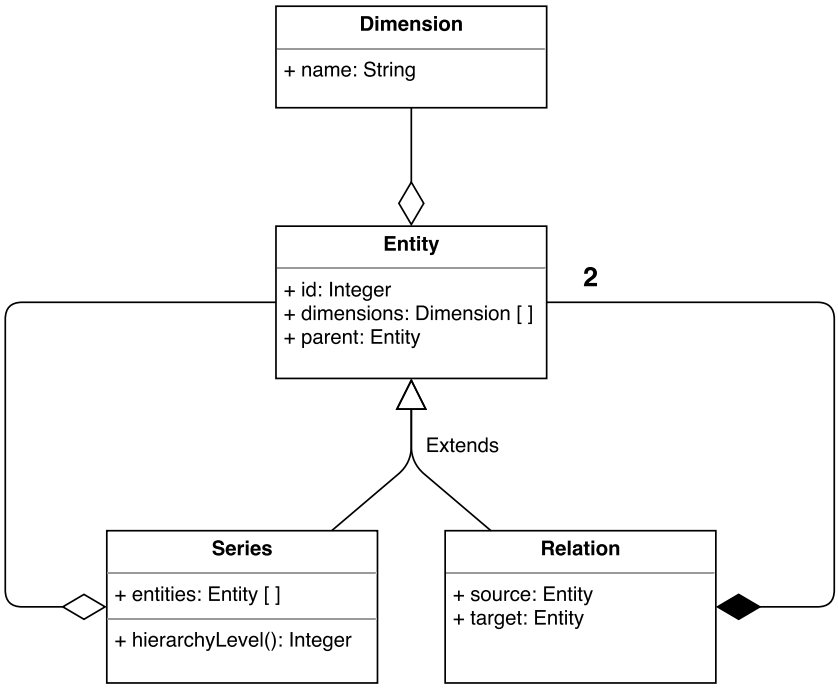


Fig. 4.1: A data structure for tabular, hierarchical and relational data

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

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

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

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

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

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

$$\mathbb{V} = \{x, y, y2, y3 \dots yn, z, height, colour, size, shape, orientation \dots\} \quad (4.6)$$

A discrete set to capture visual variables position, size, orientation, texture, colour and shape, as described in Section 2.3.1. It is required to allow for *encoding* interactions, that change the mapping of a dimension to a visual variable.

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

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

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

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

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

$$Filter : \mathbb{E} \to \{\bot, \top\} \quad (4.8)$$

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

$$Window : \mathbb{D} \to \mathcal{P}(Space(\mathbb{D})) \quad (4.9)$$

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

The subset can be defined implicitly or explicitly: For charts with continuous values along the x- and y-axes, the function returns the representatives `from`

and `to`. These representatives implicitly define the interval in $Space(\mathbb{D})$. If a dimension has discrete values, this function can also return an explicit set of values.

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

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

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

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

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

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

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

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

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

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

4.4 Communication pattern

In Section 4.3 we discussed how we can describe the categories of the interaction. The category 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 messages are exchanged in the conceptual framework?

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

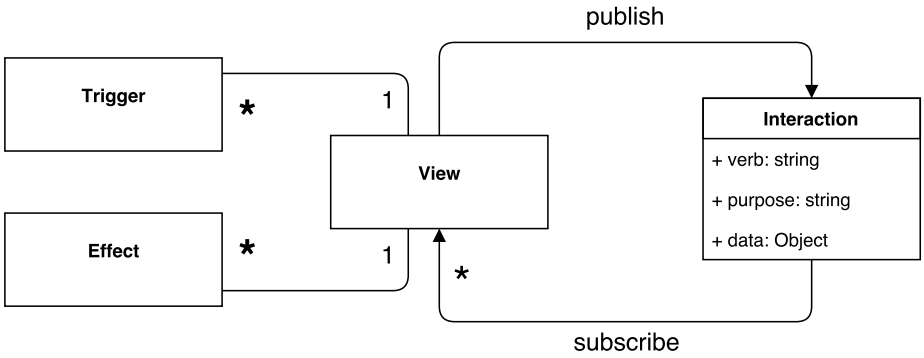


Fig. 4.2: The message flow of the communication pattern: A view can have multiple triggers and effects for one interaction. Likewise, an interaction can be notified via subscriptions to many views.

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 categories. In order trigger an interaction, the visualization simply publishes to the named interaction.

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

5

Implementation

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

5.1 Implemented interactions

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

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

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

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

If we still have time, add filter interaction here

5.2 Overview of the Layout

Figure 5.1 shows the layout of the different views from the user perspective. The 2.5D tree map is on the left while the geographic visualization is on the right. Some additional views like the message log and a form to manually change the highlighted item are visible as well.

The figure shows a screenshot of the VISUAL ANALYTICS PLATFORM, currently visualizing the data set "Wahlkreise". The 2.5D tree map is tiled based on unemployment rate and the colour is based on the percentage of high-school graduates. Currently the district "Magdeburg" is selected in the geographic visualization with an unusually high percentage of graduates for its unemployment rate.

5.3 Architecture

Figure 5.2 shows classes and their relations in the coordinated multiple view implementation of 2.5D tree map and geographic visualization.

`UAController` is a class of the existing VISUAL ANALYTICS PLATFORM and controls the rendering of the 2.5D tree map. The rendering of the geographical visualization is controlled by the class `MapComponent`.

Classes with a `render` method are views and can be rendered in place of a node inside the DOM tree. Those views are implemented as React components and therefore have a native update mechanism, i.e. they get automatically re-rendered if their internal state changes. Except of a dependency to a

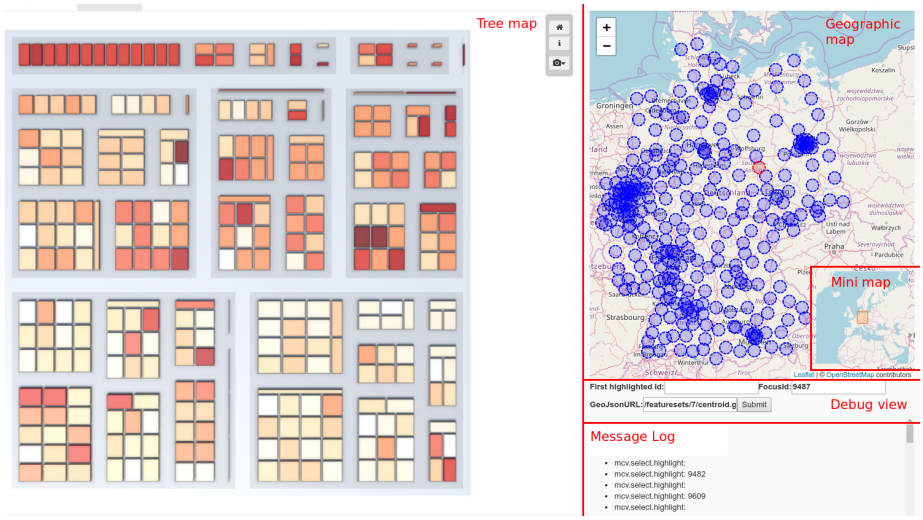


Fig. 5.1: This Figure shows the layout of the coordinated multiple view, red lines indicate the borders of different views.

`MultiviewCoordinator`, they are dependency free, which makes those classes easy to test.

Additionally, there are two other views, `DebugView` and `MessageLog`. These views record interactions at runtime and can trigger and simulate an interaction.

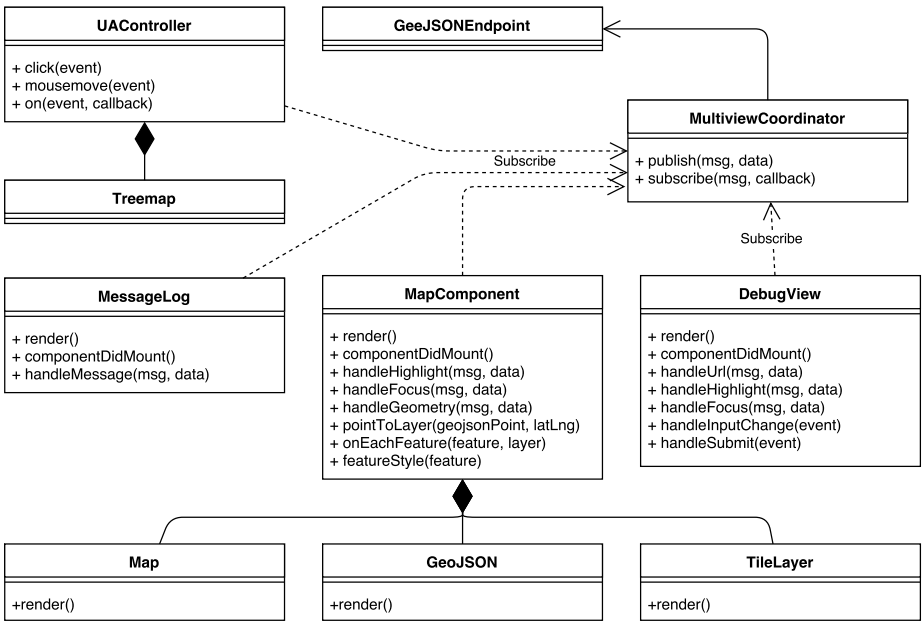
State and changes shared between views are controlled by the `MultiviewCoordinator`. Every view has a reference to `MultiviewCoordinator` in order to subscribe to interactions. The `MultiviewCoordinator` itself does not have a visual representation.

5.4 Adding the views to the DOM

Adding the geographic visualization to the DOM is straightforward. Listing 5.1 shows an example how to use React's low-level API to render components inside the DOM. It uses an id `multiview-map-component` to reference a particular node in the HTML document in Listing 5.2 on line 3. On lines 4 to 6 you can see the required JavaScript imports, i.e. two imports required by React and the compiled JavaScript application, which is called `example.js`.

Listing 5.1: Example application written in TypeScript. Views can be added to the DOM individually. The implementation exposes convenient TypeScript declarations, here `MultiviewCoordinator` and `MapComponent` are imported.

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




```
import * as React from "react";
import * as ReactDOM from "react-dom";
import { MapComponent, MultiviewCoordinator} from 'urban-
    analytics-multiview-map-component';

let controller = new MultiviewCoordinator();


ReactDOM.render(
  <MapComponent controller={controller}/>,
  document.getElementById('multiview-map-component')
);
```

Listing 5.2: This short HTML document shows how to render Views at a certain node inside the DOM. The id multiview-map-component is used as anchor.

```
<html>
    <body>
        <div id="multiview-map-component"></div>
        <script type="text/javascript" src="https://unpkg.com/
    react/umd/react.development.js"></script>
        <script type="text/javascript" src="https://unpkg.com/
    react-dom/umd/react-dom.development.js"></script>
        <script type="text/javascript" src="example.js"></
    script>
```

```
7     </body>
8 </html>
```

The reference implementation of the conceptual framework is written in `TypeScript`[1]. The implementation exports typescript declaration and you can see an import of the classes `MapComponent` and `MultiviewCoordinator` on lines 1 to 3 in Listing 5.1.

Event handlers just have to publish an interaction, if the view is subscribed to the same interaction, it will automatically re-render.

5.5 MultiviewCoordinator

The `MultiviewCoordinator` uses the `publish-subscribe` pattern for coordination and works as a broker for interactions across coordinated multiple views. In order to accomplish that it uses the JavaScript library `PubSubJS` which is an topic-based, asynchronous implementation of the pattern.

Geometry data is exchanged with `GeoJSON`[2]as data format.

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

Listing 5.3: The `multiviewController` avoids unnecessary requests to reload geometries

```
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)
       => void) {
7      PubSub.subscribe(msg, callback);
8    }
9
10   public publish(msg:string, data:any) {
```

[1] TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It provides optional static typing, classes and interfaces and helps to reduce errors by raising type errors at compile time.

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

```
11      if (msg === 'mcv.reconfigure.url'){
12        if((this._geojsonUrl == null) || (this._geojsonUrl !==
        data)){
13          this._geojsonUrl = data;
14          fetch(data, {
15            credentials: "same-origin"
16          }).then((resp) => resp.json()).then((response) => {
17            PubSub.publish('mcv.reconfigure.geometry', response);
18            PubSub.publish(msg, data);
19          }).catch((err) => {
20            console.log(err);
21          })
22        }
23      } else {
24        PubSub.publish(msg, data);
25      }
26    }
27
28    public clearAllSubscriptions() {
29      PubSub.clearAllSubscriptions();
30    }
31  }
```

As shown in Listing 5.3, 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 5.4. 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.

Listing 5.4: A truncated example of a user distribution across German federal states

```
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       },
```

```
      "id": 0
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [...]
      },
      "properties": {
        "NAME_1": "Bayern",
        "state_code": "BY",
        "user_count_total": "36",
        "user_count_normalized": "0.10746268656716418"
      },
      "id": 1
    }
  ]
}
```

5.6 Software patterns

Figure 5.3 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 `MultiviewCoordinator`. Any change to the `MultiviewCoordinator` 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.

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

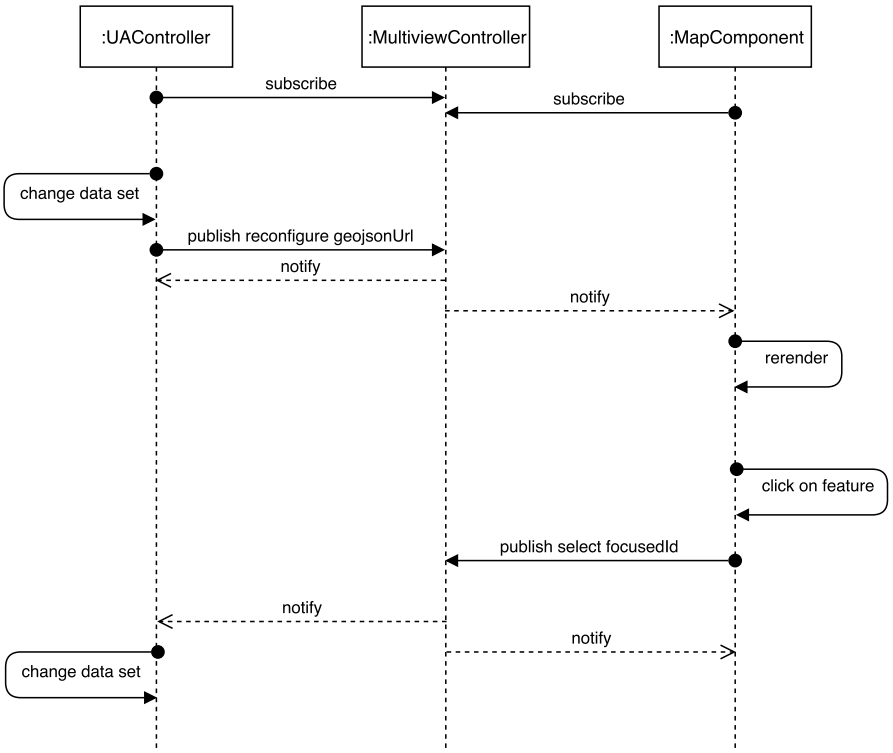


Fig. 5.3: The sequence diagram shows the notification of different components. The user first chooses a feature set and clicks on a polygon in the geographic visualization.

their closest common ancestor is the coordinated multiple view itself, controlling state and passing user interaction down to it's 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 to all dependent views. These views are then re-rendered.

Examples for the kind of data that might trigger a re-rendering of a view:

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

5.7 Map Component

The geographical visualization is implemented as a `Map` component of the React Leaflet library. Listing 5.5 shows the `render` method of the component.

Listing 5.5: `render` method of the Map component of the geographical visualization

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

The `render` method is the only required method of a React component. It will be invoked on the initial rendering of the component of the `DOM` and on every update of the component's properties. React's templating language "`JSX`" allows to nest other child components into the React parent component. In this case the `Map` component includes a `TileLayer GeoJSON` component from the `react-leaflet` library. This library conveniently provides "React components for Leaflet maps." [9].

5.7.1 GeoJSON Component

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

This way, the parent `Map` component controls the data flow and without a `geojson` object, no polygons are placed on the map. A changed `geojsonURL` will always update the child component as it is used a `key` on the `GeoJSON` component. The callbacks passed into the `GeoJSON` component control the visual representation of each polygon and they add event handlers for a mouse click or a mouse move on each polygon. Listing 5.6 shows the event handlers added to the map.

Listing 5.6: `onEachFeature` callback, adding handlers for mouse events

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

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

Listing 5.7: `featureStyle` callback, configuring the visual appearance depending on the currently highlighted or focused feature ids

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

```
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  }
```

The `featureStyle` in Listing 5.7 method is very straightforward. If the feature is currently focused, the `fillColor` of the polygon is red, otherwise blue. Likewise, if the feature is currently highlighted, the polygon has a white, solid stroke.

Listing 5.8: `pointToLayer` callback, if a feature of `GeoJSON` has a point geometry, it will be shown as a circle

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

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

6

Evaluation and Discussion

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

6.1 Use Cases and Discussion

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".

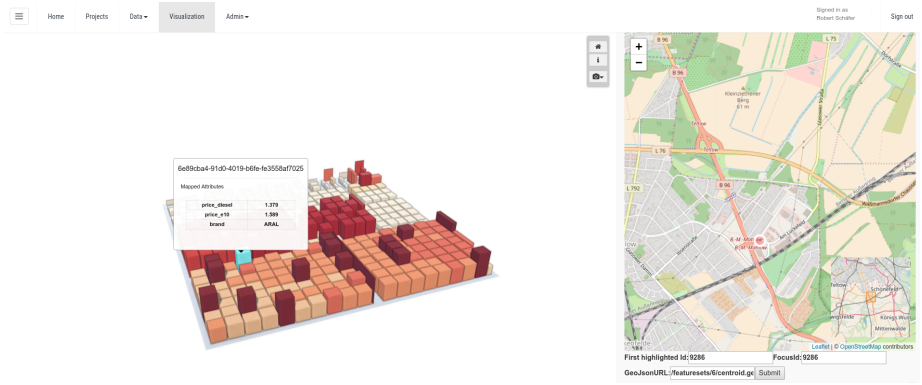


Fig. 6.1: The 2.5D tree map shows gas stations with a layout based on the brand name and height and colour mapped to price. Relatively expensive gas stations, with respect to the other gas stations of that brand, are often located next to highways.

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.

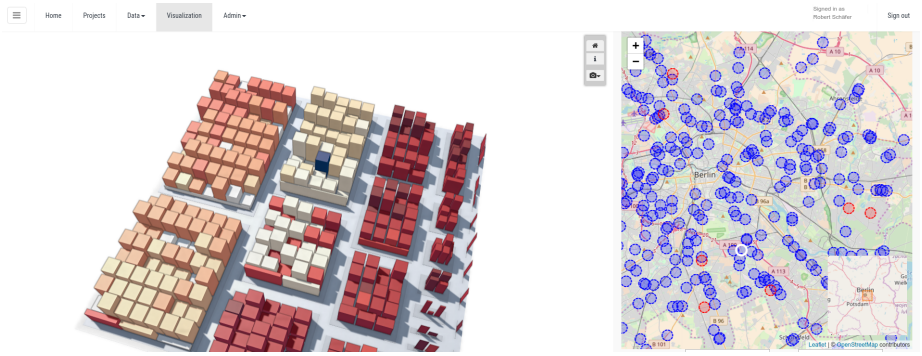


Fig. 6.2: A multi select allows to collect a group of interesting features.

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.

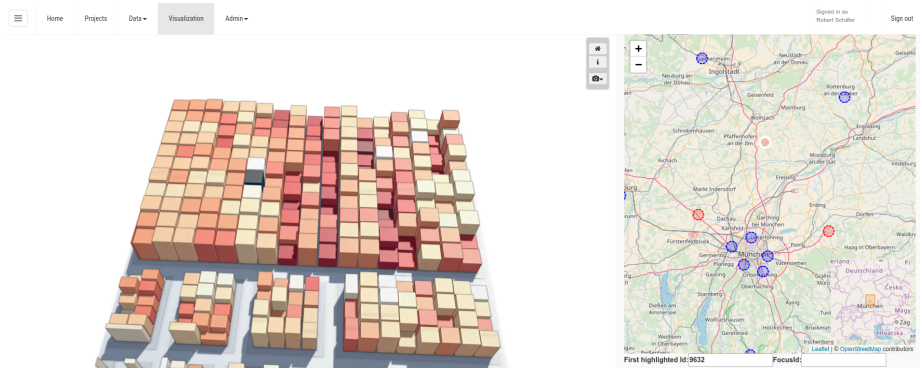


Fig. 6.3: Three unsuspicious districts happen to be located next to Munich. Apparently people move out of Munich to avoid high rents.

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.

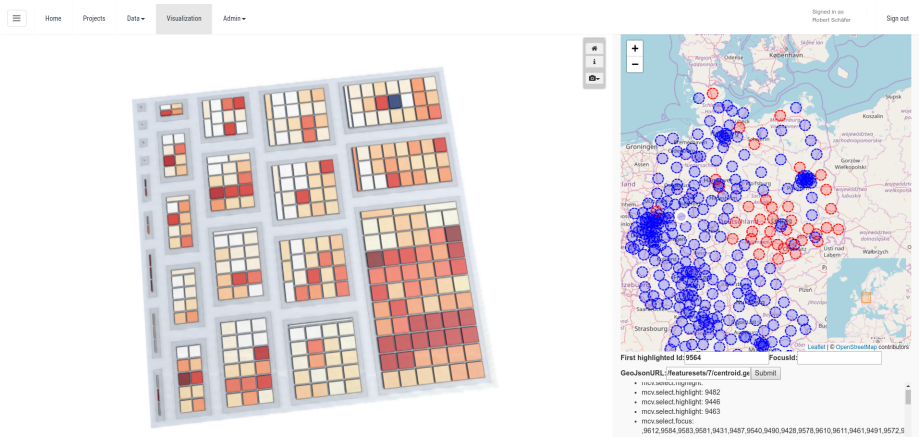


Fig. 6.4: The layout of the 2.5D tree map is based on area of the district, the colour based on the unemployment rate. We did a multi select of all rural districts with a high unemployment rate. We can see a strong correlation with a location in the east of Germany.

6.2 Performance Evaluation and Limitations

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

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.

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

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.

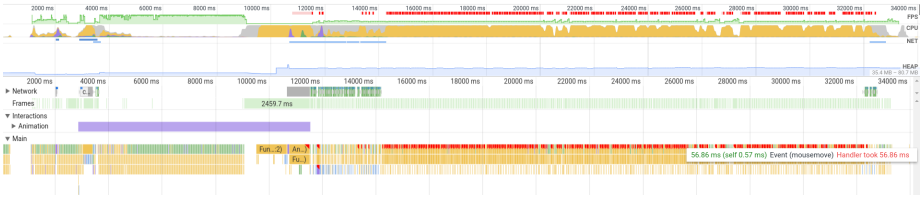


Fig. 6.5: During profiling of the “Immoscout” data set, only 16 frames per second get rendered

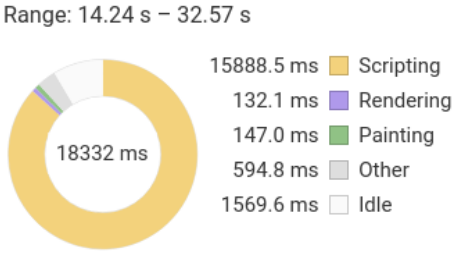


Fig. 6.6: Visualizing just the 2.5D tree map, almost the entire CPU time is spent during Scripting

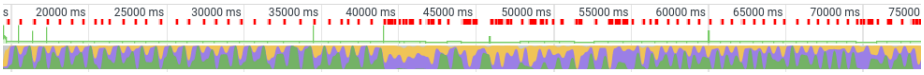


Fig. 6.7: Every highlighting interaction comes with a spike of CPU time for rendering and painting. Focusing on a feature redraws the background and leads to an increase of painting time.

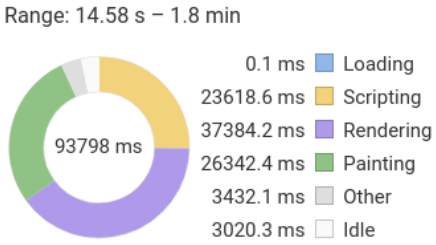


Fig. 6.8: The geographical visualization spends much more time during re-rendering and painting.

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.

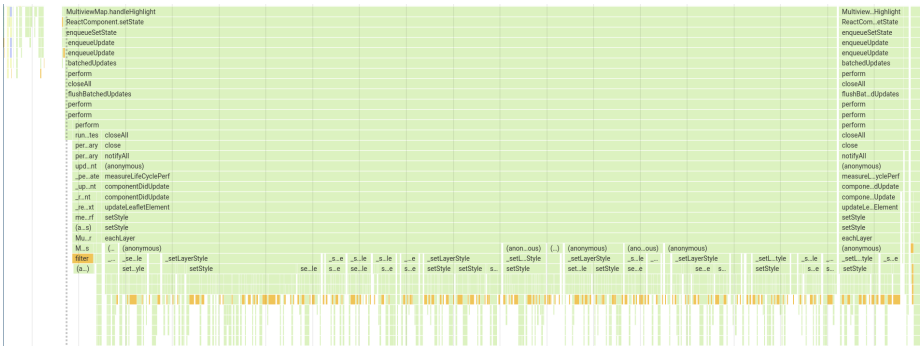


Fig. 6.9: Event handler `handleHighlight` takes about 90ms to finish. During this time, the list of features is iterated and `setStyle` executed.

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.

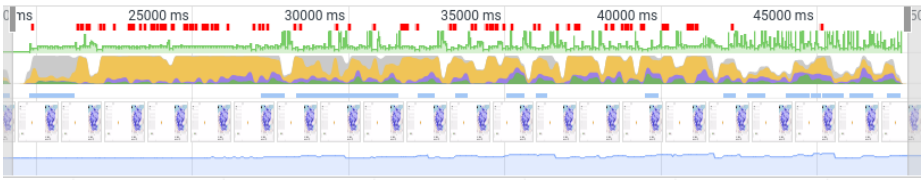


Fig. 6.10: The number of features is crucial for the time spent during a highlighting interaction.

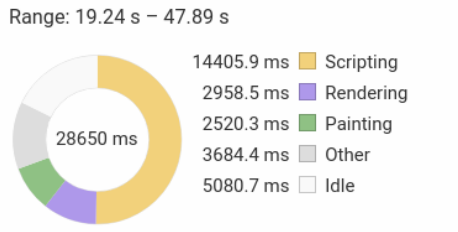


Fig. 6.11: Data set "Landkreise" shows less time spent on painting and rendering.

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.

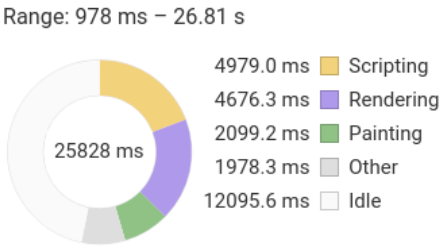


Fig. 6.13: Without a 2.5D tree map the CPU is idling for almost 50% of the time.

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 visualizations keeps a record of every past interaction and can replay the record up to the desired step (2) or the mathematical inverse of the function is published. Both options have an undesired impact on the required memory, because data needs to be kept in memory for every visualization. If the record of interactions is long, the first implementation will take a long time to replay every interaction up to the desired state.

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

Data extensibility of interactions is enabled with arbitrary JavaScript objects. In Section 3.3 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.

7

Summary and Conclusion

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

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

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

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

7.1 Future Work

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

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

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

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

Did we achieve our goals?

Is the concept sane in regarding the implementation?

List stuff which was not accomplished in this master thesis

References

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

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

[6] 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.

[7] 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.

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

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

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

[11] 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).

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

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

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

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

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

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

[24] 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.

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

[26] 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.

[27] 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).

[28] 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.

[29] 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).

[30] 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.

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

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

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

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

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

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

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

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

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

[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.