FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Semantically Zoomable Choropleth Map

Data visualization on several administrative levels

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Lucas Dworschak
Registration Number 1225883

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Ass. Dr.techn. Manuela Waldner, MSc

Vienna, 7th September, 2016

Lucas Dworschak	Manuela Waldner

Erklärung zur Verfassung der Arbeit

Lucas Dworschak
Mühlgasse 16/1/14, A-2481 Achau

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 7. September 2016

Lucas Dworschak

Kurzfassung

Choropleth Karten, und andere geographische Visualisierungen, werden dazu verwendet um quantitative Daten auf geographischen Regionen abzubilden. In dieser Arbeit wurde eine Choropleth Karte implementiert um zu visualisieren wie viele wissenschaftliche Arbeiten von verschiedenen administrativen Regionen publiziert wurden. Der Hauptunterschied, zu herkömmlichen Choropleth Karten, ist, dass eine Navigation der Karte möglich ist. Der User kann die Karte zoomen und den Kartenausschnitt verschieben. Was diese Karte so speziell macht ist die Verwendung von semantischem Zoom. Semantisches Zoomen erlaubt es dem User den Detailgrad der Karte zu verändern. Das Ändern des Detailgrades bedeutet, dass sich die Karte auf bestimmten diskreten Zoomstufen verändert und zum Beispiel eine administrative Region in mehrere kleinere unterteilt. Diese kleineren administrativen geographischen Regionen werden dann wieder eingefärbt, wodurch eine neue detailiertere Choropleth Karte entsteht. Zusätzlich wurden noch weitere Interaktionen mit der Karte umgesetzt. So kann der User auch noch die Darstellung der Karte beeinflussen aber auch den dargestellten Datensatz auf bestimmte Attribute filtern.

Abstract

Geographic visualizations, like choropleth maps, are used to visualize data on geographic regions. In this thesis a choropleth map was implemented to display quantities of publications of scientific texts and papers. With the use of a choropleth map the viewer is able to interpret how quantitative data changes on different geographic regions. The main feature that distinguishes the implemented choropleth map from conventional ones is the use of map navigation. The choropleth map can be zoomed and panned to different map regions. What makes this map navigation so special is the use of semantic zooming to allow the level of detail of the map to change on discrete zoom steps. The change of the level of detail means that administrative regions are being divided into smaller administrative regions which are than again colorized individually to create a new, more detailed, choropleth map. Other interactions with the choropleth map are introduced additionally. The other interactions with the map range from the manipulation of the map appearance to filtering the displayed data set.

Contents

CHAPTER 1

Introduction

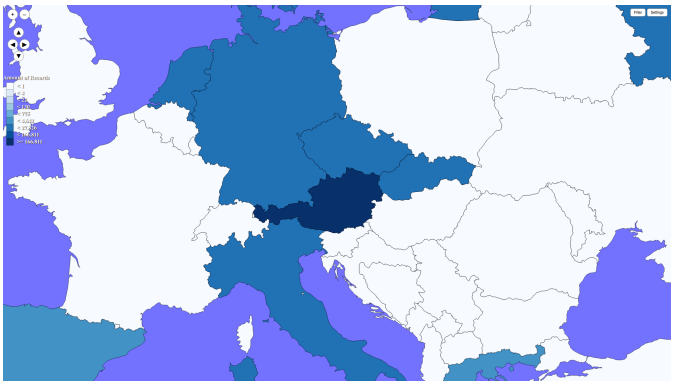
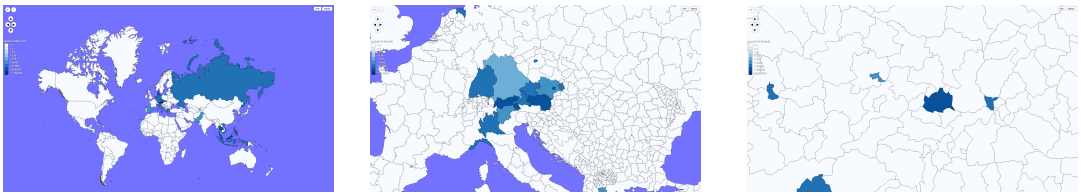


Figure 1.1: Choropleth map of Europe.

Choropleth maps (see Figure 1.1) are used to visualize quantitative data on geographical regions. Most choropleth maps used to date can only visualize data with little to no interactions from the user. There are some choropleth maps where the user is able to zoom in so that he can get a better look at smaller geographical regions. However, conventional choropleth maps do not give the user more control over the navigation of the map.

The main feature of the implemented Semantically Zoomable Choropleth Map (SeCho map), is the advanced use of map navigations that allow the user more control over the visualization of the data. Using these advanced map navigations, the user is able to change the region he is viewing using panning or zooming (see Figure 1.2). With the lowest zooming level the user can view earth as a whole (see Figure 1.2a). When the user zooms into the map, semantic zooming is used to increase the level of detail of the map (see Figure 1.2b and 1.2c). While ordinary choropleth maps would only display one

administrative level, the SeCho map allows the visualization of two or more levels of administrative division.



(a) Administration level 0 (b) Administration level 1 (c) Administration level 2

Figure 1.2: Difference of semantic zoom levels.

The SeCho map can be implemented for any kind of data. In this case, the map was created for the Centre for Social Innovation (ZSI) [ZSI] to display publication quantities of scientific papers and texts. The ZSI already provided a database with those records and their relations to authors and institutions. The database focuses mostly on publications of Austrian research institutions. The dataset additionally allowed the use of filters on bibliographical meta data and the visualization of co-publications to show scientific relations between geographic regions. Both features were added to the implemented SeCho map.

CHAPTER 2

Related Work

Before the concept of the SeCho map will be detailed, a summary of visualizations and technologies will be described below. In the first section other geographic visualizations will be introduced and their advantages and disadvantages will be discussed. The second section will describe technologies and concepts that are being used in order to visualize the SeCho map.

2.1 Geographic Visualization Techniques

2.1.1 Choropleth Map

Choropleth maps [FBC00] visualize data by coloring each administrative division. It isn't necessary to know the concrete coordinates of the values which are visualized. An assignment to an administrative division is all it needs to represent the data. But by displaying the dataset this way, the actual data might be greatly misrepresented, by visualizing that a whole country contributed to the value evenly, while actually only one particular location was responsible. Different types of color schemes can be used depending on the type of data. While color schemes that use nominal colors are best suited for visualizing nominal data (e.g. classifications), quantitative data is most likely represented by a sequential color scheme. Additionally it is also possible to visualize data using a diverging color scheme. This color scheme is used for data attributes, that split the data into two groups. It is mostly used for ordinal data, which splits the data at an average value into a group above and a group below this value. Since the data used in the implemented SeCho map is quantitative, a sequential color scheme was used. The mapping of quantitative data to a sequential color scheme can be further defined by the used scale. For example a logarithmic scale should be used, if the dataset consists of many lower quantitative values and a few very high outliers. If a linear scale was used for this kind of dataset, the lower values could most likely not be distinguished since they would be mapped to the same color group.

2.1.2 Necklace Map

A necklace map [V+10] colors administrative divisions of a map with a nominal color scale. Each administrative division is matched with a symbol with the same color. The symbol visualizes the quantitative data with its size. Each symbol is located on a curve, a so called necklace, that curves around the displayed map. The distance between administrative division and its associated symbol should be as minimal as possible without obstructing the visualization of the map. The advantage to using this kind of geographic representation of data is that the size of each country doesn't matter. The user can look at the map and easily see where the maximum and minimum is. Furthermore all symbols can be compared without error. The disadvantages of this method are that the symbols and necklaces create a more cluttered view, and the assignment of symbols to administrative division is not too intuitive. Furthermore for interactive maps, the placement of each necklace would have to be recalculated every time the user pans or zooms the map.

2.1.3 Heatmap

Heatmaps [SMKH09] are generated by colorizing spots of the map where data exists. The more quantitative data exists on one spot of the map, the brighter the spot gets. This allows the user to see with a glance where hot spots are located on the map. One disadvantage using this approach is that data cannot be grouped according to geographic units such as countries or other administrative divisions. This means when the user looks on the map he can see the regions where most of the quantitative data is. But he cannot differentiate to which administrative division it belongs and correctly see the total amount an administrative division has.

2.1.4 Proportional Symbol Map

Proportional Symbol Maps [CHVKS10] visualize quantitative data by rendering symbols with different sizes on the map. If a symbol is larger than another one, displayed on the map, it represents a larger quantitative value. The location of each symbol mostly represents the exact longitude and latitude coordinates of the visualized data. But it is also possible to display proportional symbols on a grid or the center of an administrative division.

2.1.5 Dotmaps

Dotmaps [DBBCM04] represent data by visualizing dots of equal size. Depending on the dataset of the map, one dot may represent singular occurrences or large quantitative values. An example for a singular occurrence may be cases of infections in a city. An example for large quantitative values may be population numbers, where one dot represents for example 100,000 people. The location for singular occurrence is trivial most of the time, but it isn't for large quantitative values. For those values the dot representing the data is often rendered at the mean coordinate of all values it represents.

2.1.6 Cartogram

A cartogram [Rai34] visualizes quantitative data using the size of each administrative division. In order to prevent creation of holes between administrative divisions, each division is distorted to accommodate the scaling of neighbor divisions. The main disadvantage resulting from this is that the map can be distorted beyond recognition. Furthermore because they are no uniform shapes, the values of each administrative divisions cannot be appropriately compared to each other.

2.1.7 Isopleth Map

An isopleth map [OK10] allows the visualization of continuous distribution of quantitative data. It is not limited to borders of administrative divisions and shows smooth transitions of quantitative values on the map. Isopleth maps can be created by laying a discrete grid above the map and calculating the quantitative value of each grid cell. Unlike visualizations like the proportional symbols map, the isopleth map can group areas with similar values. Unlike the choropleth map or cartogram it can only really be used for data with concrete coordinates.

2.2 Used Map Algorithms

2.2.1 Semantic Zooming

Semantic zooming [PF93] is a technique that can be used to change the level of detail of a visualization. When the data is displayed from a higher zoom level, the level of detail of the data is quite low. When someone zooms in, more detail about the underlying data is revealed on certain zoom steps. By using this method the user can get an overview over the approximate distribution of data and can later choose to visualize certain data with more detail.

2.2.2 Maptiles

Maptiles [MMM14] are used in order to separate a map into a uniform grid. This allows a separation of data into discrete cells called tiles. Since most of the time only a small part of the map is visible, the map application has to know what data it has to load from the server. The application knows which cells are currently visible in the viewport, and each cell stores which information it holds. Therefore the application can use only content that is stored in each visible cell and display only those to the user. The SeCho map uses maptiles specifically to store information about the underlying geographic data. It enables an efficient way to retrieve administrative divisions which are located within each cell and allows the application to cache this information to use it again at a later time.

CHAPTER 3

Semantically Zoomable Choropleth Map

The Semantically Zoomable Choropleth Map (SeCho map) allows the user to compare quantity data between administrative divisions. Administrative division are established by governments to divide the world into smaller regions, e.g. countries, states, etc. Subdivisions of administrative divisions are done hierarchical. Each level of subdivision is called an administrative level. For example Austria can be divided into five administrative levels: The country itself is divided into 9 states, every state can further be divided into multiple districts and every district can be divided into municipalities and cadastral municipalities. In a normal choropleth map, the creator of the choropleth map defines which administration level is displayed to the viewer. In the SeCho map, the user can use zooming and panning to define the visible area of the map. Furthermore the SeCho map uses semantic zooming [PF93] to change the administration level on discrete zooming steps to display the map with an appropriate level of detail.

The data set used in the implemented SeCho map are publications of scientific papers and texts provided by the Centre for Social Innovation (ZSI) [ZSI]. The data set is stored in a database, which stores bibliographic meta data for each publication. The quantitative value, which represents how many scientific papers a given administrative division has published, is calculated on the server at runtime. The server simply returns a tuple of a numeric quantitative value and a geo-reference to correctly map the quantitative value to the correct administrative division. Administrative divisions are stored and visualized to the user as polygons. Each polygon is drawn with a certain fill color of a discrete color scheme to visualize the quantitative value.

Since the SeCho map can display regions from any part of the world on multiple administrative levels, maptiles [MMM14] were used to identify the currently visible parts of the map. Furthermore maptiles are also used to efficiently retrieve and store data of

regions that are located within this tile. A grid of uniformly sized cells was used to divide the map into tiles. Each administration level that can be visible due to semantic zooming has its own grid with appropriately sized cells. That means that higher administration levels, which are visible when zoomed in to the map, have smaller cells than lower administration levels.

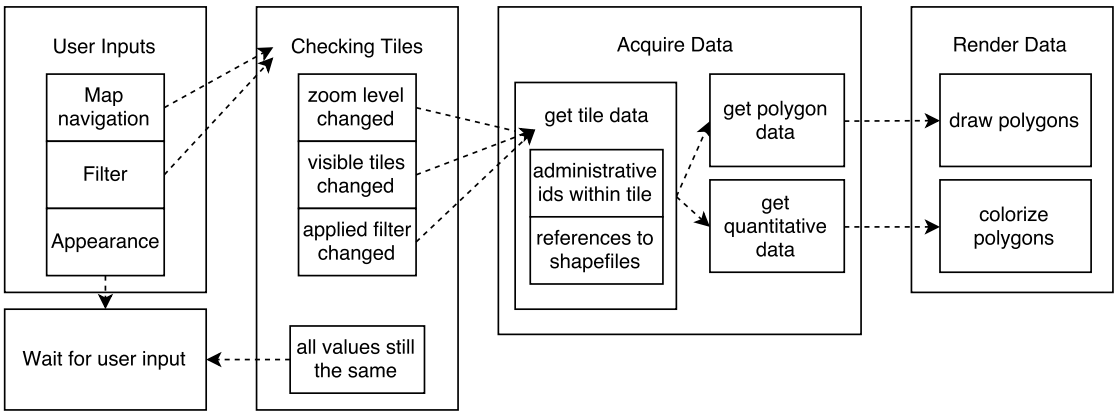


Figure 3.1: Pipeline of the SeCho map.

As you can see in Figure 3.1, the main parts of the SeCho map are divided into:

- Detection and interpretation of user input,
- determining whether or not the state of the map changed,
- acquisition of data, and
- rendering of data.

In the following sections these main parts are being described with more detail.

3.1 User Input

The user can take several actions to define how data is being displayed on the interactive choropleth map. Performable actions include:

- Map navigations like panning and zooming,
- filtering according to data attributes, and
- changing the appearance of the data.

In the implemented SeCho map only the first two user inputs require that the next step of the pipeline seen in Figure 3.1 is executed. The changing of appearance is only applied to the map after one of the first two actions were done by the user.

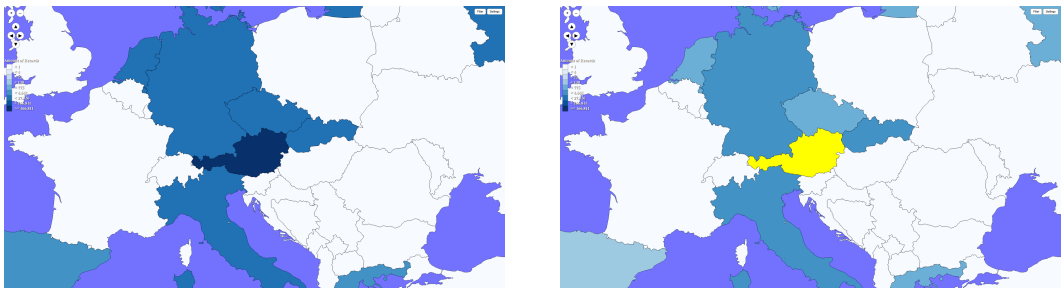
3.1.1 Map Controls

Similar to other maps with navigational control, like OpenStreetMap [OSM] or Google Maps [GMA], the user can change the area that is being displayed with panning and zooming. The panning action is done by the conventional method of dragging the map across the screen. The whole map moves in the same direction as the mouse. While new shapes become visible on one end, shapes on the other end are being hidden from the viewport. Like the panning action, the zooming action also follows the conventional method of scrolling with the mouse wheel to zoom in and out of the map. The map zooms into or out of the map and reveals/hides the shapes around the old viewport and scales the shapes accordingly.

In addition to the conventional zooming method where shapes of countries are simply scaled, the application also performs semantic zooming [PF93]. The SeCho map uses discrete zoom steps to determine when the administration level, and therefore the level of detail, changes. The appropriate zoom steps where evaluated manually and are the same for each country. In the implementation of the SeCho map, shapes from GADM.org [GAD] were used for polygonal data of each administrative division. Since they defined on what administration level what administrative division belongs to, the sizes of administrative divisions on one level varies from country to country. Therefore it is possible that one administration level displays both large and tiny divisions.

3.1.2 Filtering

The concept of SeCho map can be implemented with any kind of data set which may or may not have additional attributes that allow to specify which data is being visualized on the SeCho map. The data set used in the implemented SeCho map also stores meta data of each publication, which can be used for filtering. Filter criteria that were used include attributes like publication year, number of authors and number of citations. For other data sets the filter criteria have to be adapted accordingly.



(a) Normal choropleth of Europe.

(b) Co-publications with Austria.

Figure 3.2: Comparison between conventional map and map with co-publications.

Since some papers are created by multiple scientific institutions, the SeCho map can use a special kind of filter. An administrative division can be selected using a button located in

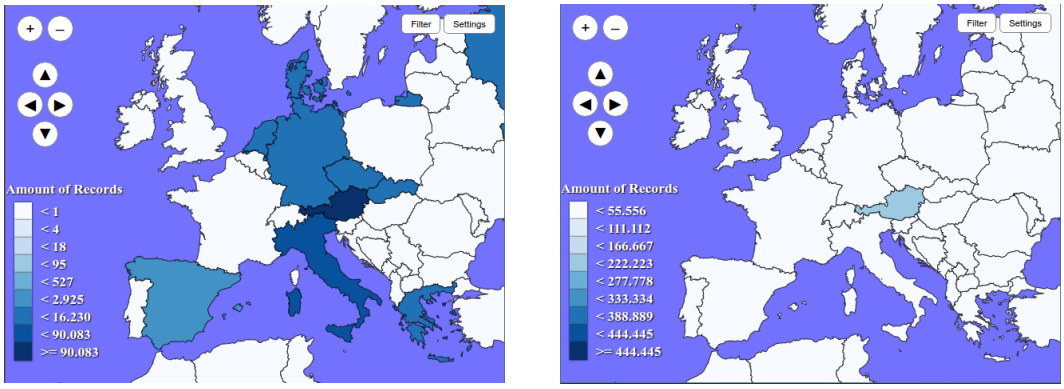
a context menu. Using this selection the SeCho map can calculate how many publications were developed with other institutions in other administrative divisions. The selected division is colored in a prominent color which is not part of the color scheme (in this case yellow) in order to differentiate between the selection and other administrative divisions. The other divisions display in color the amount of papers that have been developed with the selected unit. How the SeCho map appears to the user before and after applying the co-publication filter can be seen in Figure 3.2. Once an administrative region has been selected, the user can continue to use all user actions available to him. He can set other filter options, change the appearance of the data and continue to navigate the map fully. This includes the change of administration levels. Once an administration level has been changed the selected administrative division is no longer colored differently. This is because the lower level administrative division of the selected division may also have correlations with the selected division. For example if the state Lower Austria is selected and the user zooms out to view all countries of Europe, Austria itself can also have a correlation value if for example an institution in Vienna has co-published papers with an institution of Lower Austria

3.1.3 Appearance of Data

Another method to interact with the choropleth map is to change the way the data is presented to the user. Since there are a couple of ways to use the SeCho map, it is also important to allow the user the freedom to change the way the data is presented.

First and foremost the user is able to manipulate the scale, which splits quantitative values into discrete colors. He can either set specific values of the minimum and maximum of the scale or let the application calculate these values on runtime. Each option has advantages and disadvantages. A scale that automatically calculates the minimum and maximum of the extant might be suited best to display data of a static choropleth map. But it is not so well suited for an interactive map. Depending on the data and the current viewport of the map, a simple pan of the map can change the scale extant drastically. If a user compares two divisions that are to far apart, and would be to small to compare if zoomed out accordingly, he would have to pan the map. But since the scale extant and therefore the representing colors change while panning to a new position, the user can't just simply compare both colors. For this use case the user is able to stop the scale from changing and compare both divisions with the same scale. To combine both approaches it would also be possible to calculate the maximum displayed quantity of any administrative division, and transmit this value to the client application at initialization.

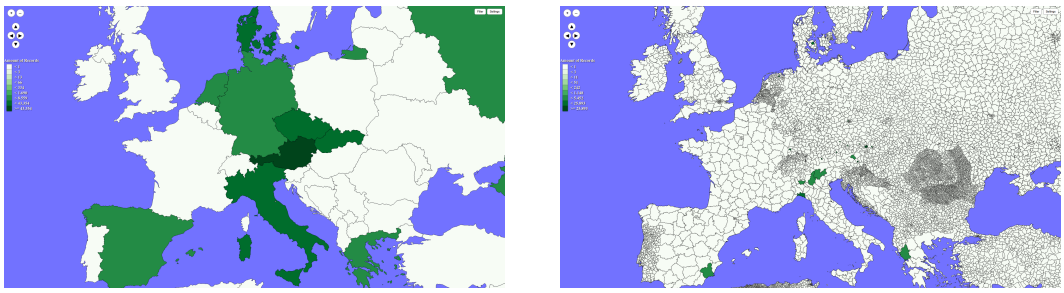
Since the used dataset consists of many administrative divisions with smaller quantities and a few with many, the choropleth map uses the natural logarithm to map values to discrete colors of a predefined color scheme. The use of the natural logarithm allows the user to differentiate divisions with smaller values more easily, and therefore achieves that more colors of the color scheme are used to visualize the data on the map. But since it might be useful to view the data differently, the user can also change to a linear scale. Figure 3.3 shows how drastic the difference between both scales are. While you can see



(a) Logarithm scale (b) Linear scale

Figure 3.3: Comparison between the linear and the logarithm scale.

on 3.3a that nearly every shape with a value is being colorized, the linear scaled map seen on Figure 3.3b only shows that Austria is colored. This is due to Austria having a much higher quantitative value in the used dataset.

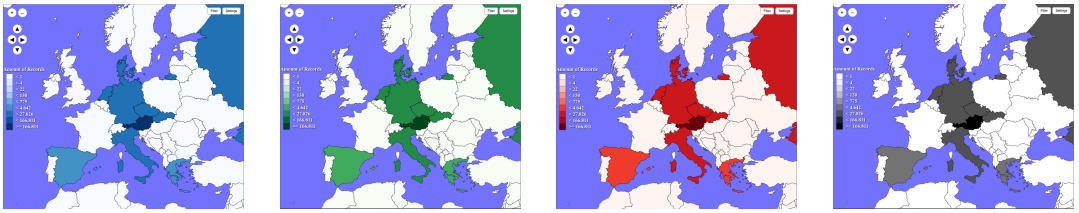


(a) Administration level 0 (b) Administration level 2 of Europe

Figure 3.4: Comparison between a map with normal semantic zooming and a map where semantic zooming was disabled.

Another way to manipulate the appearance of the map is to allow the user to stop the semantic zooming. This effect can be seen on Figure 3.4. This would allow him to display a more detailed administration level while being zoomed out further than previously possible. This method can allow the previously mentioned comparison of divisions that are farther away from each other. Unfortunately it is still limited by the size of each division.

In addition to the aforementioned methods, that truly change the way the map behaves and appears, it is also possible to change the color scheme of the map. This feature can be seen in Figure 3.5. Although it is mostly a method to accommodate user preferences it can also yield useful if the presentation medium has difficulties to differentiate between a certain set of colors (e.g. projector). In the implemented application four different



(a) Blue color scheme. (b) Green color scheme. (c) Red color scheme. (d) Black color scheme.

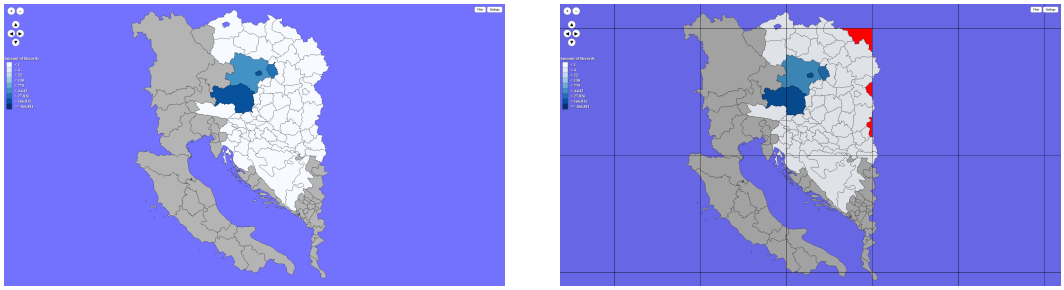
Figure 3.5: Comparison of color schemes.

color schemes are supported (blue, green, red, black). These color schemes were created using Colorbrewer 2.0 [COL].

3.2 Checking Tiles

Maptiles [MMM14] are used to allow a local caching of data, since otherwise every map navigation interaction with the map would have resulted in a new request to the server. The map is being divided into rectangles with equal longitude and latitude. Every administration level has its own grid of tiles of appropriate sizes. Each maptile holds two kinds of information. First of all each tile stores a list of references to administrative divisions which are located completely or just partially within the tile. The second information, each maptile stores, is a list of references that indicate which file, where polygon data is stored, it has to load in order to render this tile. One file, in the following called shapefiles or shapetiles, holds multiple polygons that define multiple administrative divisions. Choosing the correct size of each level's maptiles is important. On the one hand if the tiles are too big, the data that is being loaded might never be needed. On the other hand if the tiles are too small, the overhead of each network request to load each shape might be too big and increase the overall network usage unnecessarily. Additionally smaller shapetiles will load new data more frequently if the map is panned or zoomed out, which would create too much unnecessary network traffic. The difference between shapetiles and maptiles can be seen in Figure 3.6. In Figure 3.6a you can see how the contents of a shapefile looks like. In Figure 3.6b you can see that black bordered maptiles were drawn over the shapetile. The area colored red indicates regions that are not within the rendered shapetile but in a neighboring one. In order to draw everything within one maptile, the maptile has a list of references to all shapetiles that have to be rendered. A maptile has to display all shapetiles referenced within the list, in order to render everything in the maptile without holes.

After a user navigated the map, or changed the filter criteria, a method to check the state of the map is being executed. This method first checks whether or not the current administration level differs from the old one. Next it determines the bounding box of the current viewport. With the bounding box calculated, the currently visible map tiles can be determined and compared to the previously visible tiles. On the last step the current



(a) Contents of shapetiles. (b) Shapetiles with maptiles overlayed.

Figure 3.6: Comparison between maptiles and shapetiles.

applicable filter is compared to the old filter. If any of these three values differ from the old ones, only then the rest of the map render algorithm is being executed.

3.3 Acquiring Data

After the previous method determined that the map changed, the necessary data to draw the choropleth map is acquired. In order to reduce network traffic, some caching methods were implemented to store quantitative data and data within maptiles on the client side. Therefore every method described below always determines whether or not the data has already been loaded or has to be loaded from the server. In Figure 3.7 you can see all communications between server and client at runtime and what data is being transmitted between both.

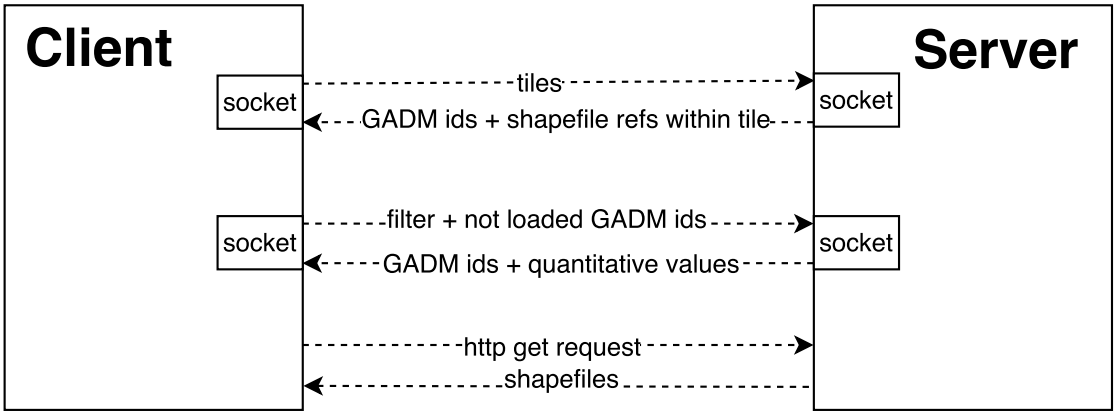


Figure 3.7: Diagram that shows what data is transmitted at runtime between client and server.

The first data, that has to be loaded, is the content of the map tiles. The server stores ids of administrative divisions with their preprocessed rectangular bounding boxes. The server is given a list of bounding boxes of each map tile, calculates which division bounding

box is at least partially in the maptile bounding box and returns a list of ids for each administrative division in the tile. The server additionally returns a list of shapefile references. These references enable the client application to know which shapefiles it has to load to display the given tile without holes in them. A shapefile simply contains all shapes within a predefined tile. The tile size of shapefiles and the tile size of maptiles are the same. Each shapefile only contains polygons of administrative divisions where the polygons center point is located within the maptile. This was done to eliminate loading the same shape multiple times. But because of this a maptile needs to know which shapefile tiles it has to load in order to display a maptile without having holes. This information is loaded from the server at runtime.

Each visible maptile now contains a list of administrative division ids and a list of shapefile references. All lists of visible maptiles now get concatenated into a list of visible ids and a list of visible shapefile references. Since each list would most likely contain multiple duplicated items, the duplicate entries are removed from both lists. Furthermore to reduce the network load it is determined which data has already been loaded from the server at a previous time, and which has to be loaded now. Only then the not loaded shapefiles and quantitative values will be loaded from the server and saved upon return on the local client cache.

3.4 Rendering of Data

There are two kinds of data that are being presented to the user, the administrative polygons and the quantitative data represented as the color of these polygons. There are two approaches to render both data on the map. Either the application draws and colorizes the shape at the same time or these steps are done separately. Since the implementation of the SeCho map allowed the separation of both steps with the use of SVG and CSS, the polygon rendering and colorization is done separately. This increases the performance of the SeCho map since it can execute one method, while it waits for the other data to finish loading. Additionally most of the time the loading of shapefiles is faster than the loading of quantitative data. Therefore the applications displays something sooner to the user than with the combined method, which creates the impression that the SeCho map waiting time for the map to finish loading is shorter.

CHAPTER 4

Implementation

The implementation of the SeCho map will be described in four main parts. First of all the data and its formatting is being discussed. The next section describes the protocol between the server and the client. In the third section the handling of the server is described. Lastly the implementation of the client application is being discussed.

4.1 Data

The two main types of data that are being used are quantitative values and the shapes of the administrative divisions. The quantitative data used for the implemented SeCho map are publications of scientific papers and texts. In the PostgreSQL [POSb] database every publication is stored in a table as a separate entry with additional bibliographical meta data. Each entry can also be assigned to an administrative division. A database query is used to group all records with the same administrative division at runtime and return the amount of records with a reference to the associated administrative division.

4.1.1 Administrative Divisions

For the shapes of the administrative divisions the world maps of GADM.org were used. GADM.org is a database for global administrative areas. It provides the shape for each administrative division on every administration level with additional meta data like name and id. The shapes of the GADM database where already used by the Centre for Social Innovation (ZSI) for mapping each research organization to the correct GADM id. Therefore it constituted the perfect solution to use the same shapes and ids to build the choropleth map and map the quantitative values to the correct shapes.

In Figure 4.1 you can see how different administration levels are structured. You can see that administrative divisions are structured hierarchically and that every division has an identifier. The identifier seen in the figure is self generated and originates from

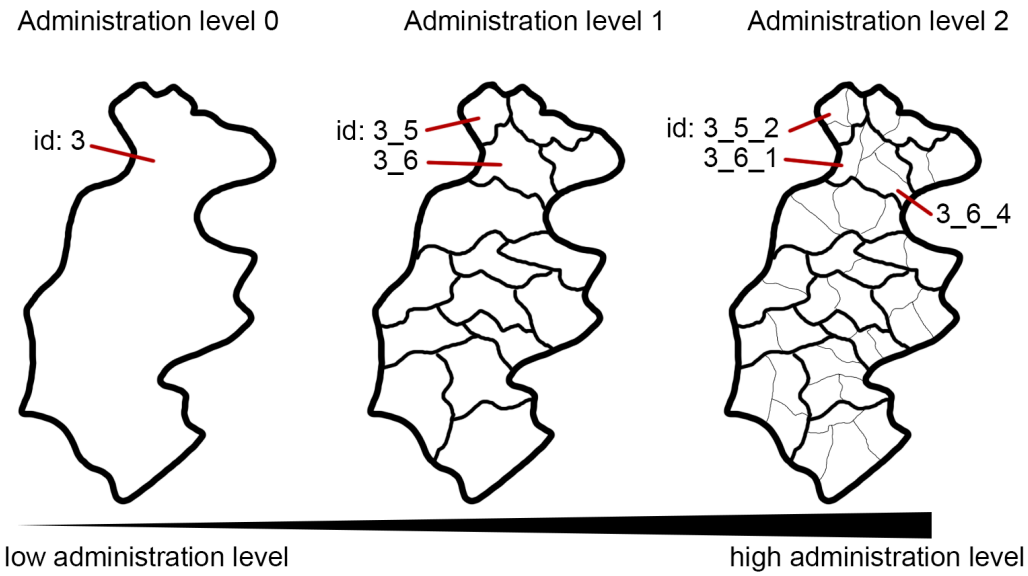


Figure 4.1: Diagram that shows the hierarchical structure of administration levels and ids of administrative divisions.

the GADM id which is an array with six integer values. Since administrative divisions are hierarchical a shape with a higher administration level (e.g. a district) can also identify with his own GADM id the id of its lower administration level (e.g. state). For example the GADM id of the country Austria is `[16,0,0,0,0,0]`, while the state Lower Austria has the GADM id `[16,3,0,0,0,0]` and the district Mödling, located in Lower Austria, has the GADM id `[16,3,31,0,0,0]`. The GADM id was changed into a simpler format to allow an easier way to reference those administrative divisions. The simplified GADM ids used in the SeCho map joined the values of the array with an underscore so that they were represented by a string. If the administrative division was only on the second administration level, only the first two values were joined (e.g. `[16,3,0,0,0,0]` was changed to `16_3`). If a country only has two administration values, but the third administration level was being accessed zeros were appended to the string (e.g. `[128,1,0,0,0,0]` was changed to `128_1_0`).

All shapes were already stored with their meta data on the PostgreSQL Database. The PostgreSQL Database already provided the PostGIS extension [POSa] that allows the execution of spatial queries on the database. It would have been possible to query all polygons of the visible administrative divisions from the database and return them to the user. But since this would mean that the server would have to generate those on runtime, which would unnecessarily keep the server busy, the shapes used in the SeCho map were preprocessed and stored in static shapefiles.

In order to get all GADM ids within a given viewport, a new table with bounding

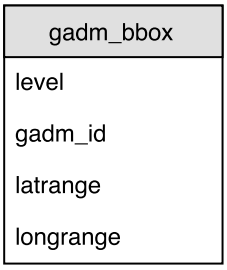


Figure 4.2: Table that stores bounding boxes of administrative divisions.

boxes for each shape was created (see Figure 4.2). The bounding boxes were stored as `numrange` fields, which allowed to store the min and max longitude/latitude points of any administrative division as ranges of longitudes and latitudes. These bounding boxes may provide some erroneous results for countries that own far away islands. But those countries aren't too many. Nevertheless bounding boxes still provide the quickest way to retrieve GADM ids that are located within another bounding box.

In order to store and retrieve the actual polygons that will be used to render the map, preprocessed static shapefiles were used. The command line tool MapShaper [MAPa] was used to simplify the polygons and change the map and its meta data to fit the needs of the SeCho map. Additionally MapShaper also converted the shapefiles into their final TopoJSON [TOP] format. In the first preprocessing step the GADM ids provided by the GADM shapefiles were changed into the previously discussed string format with underscores. Using MapShapers simplify method, the number of vertices that describe the administrative divisions were reduced to a minimum. A compromise is being evaluated, where enough vertices are being removed to minimize the file size of the map, while maintaining the overall shape of each administrative division. To do this the `visvalingam` [VW13, MAPb] algorithm was used, which tries to remove vertices first that form smaller angles with their neighboring vertices. In addition, a percent value is given to the algorithm that determines how many vertices remain after the algorithm was applied. The percentage changes depending on the administration level. For the first administration level (0) the percentage used was 0.3%. For administration level 1 and 2 the percentages for remaining vertices given to the algorithm were 0.7% and 2%. Additionally for administration level 0, which represent administration divisions on country level, small islands belonging to larger countries were removed with the `filter-islands` method. In Figure 4.3 you can see that the simplification removed most of the smaller changes within the polygon while main features and the overall shape were maintained.

The final file format used for the shapefiles is TopoJSON [TOP], an extension of GeoJSON [GEO]. Both file formats use JSON to store polygons and their attributes. TopoJSON is an optimization of GeoJSON. It only stores the edges between administrative divisions, called arcs, in a separate attribute instead of with the shape itself. The administrative division only references the arcs it needs. This prevents the need to store arcs located

(a) Before shape simplification. (b) After shape simplification.

Figure 4.3: Croatia before and after simplifications.

between two administrative divisions twice and decreases the file size by up to 80% [TOP].

The finished world map has, with the use of TopoJSON and the previously mentioned simplifications, reduced its file size to less than 1000 KB. Since the file size is small enough, the first administration level is saved as a whole without further preprocessing. The other administration levels that are being used, would still be too big to transmit to the user in one piece and have to be separated into smaller tiles. The only problem resulting from this separation is that the advantage of TopoJSON cannot be utilized to its fullest because arcs on the border of each tiles will be stored twice. Nonetheless since normally only a small portion of more detailed administration levels is being rendered, using shapefile tiles is still more efficient.

Although MapShaper also has got a method to separate a file into multiple smaller tiles, it didn't provide an easy way to know which tile contains a certain shape. To conquer this problem the simplified shapefiles where temporarily stored on the PostgreSQL database. Using PostGIS and a self written Bash script, tiles with equal length and height where created. Each tile consists of all shapes, where the center of each shape is located within a bounding box. PostGIS' `ST_Centroid` function was used to determine the center of administrative divisions and the function `ST_MakeEnvelope` was used to create the bounding box. Using PostGIS' `&&` operator both elements were used to determine whether or not the center was located in the bounding box. Furthermore a new database table was created to store a reference to the tile the administrative division is located in. How the database table is structured can be seen in Figure 4.4. In the next step, every shape of the tile is being queried from the database and a TopoJSON file was created with these shapes by using MapShaper. Additionally attributes for longitude and latitude start and administration level where stored in the TopoJSON file. These values will be later used by the client application to cache and render the tile correctly. Finally the TopoJSON file was stored in the folder for map data on the server with a predefined naming scheme. The naming scheme used was `adm<level>_<start`

`longitude>_<start latitude>.json`

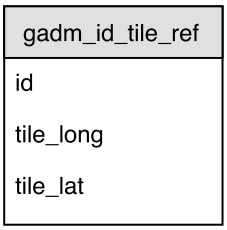
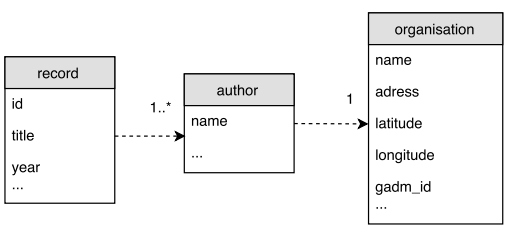


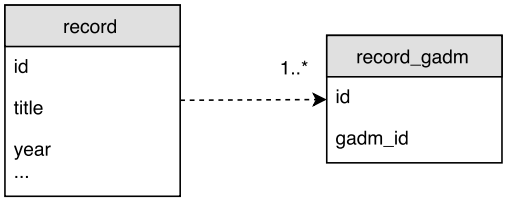
Figure 4.4: Table that stores a reference to a shapetile in which the administrative division is stored.

If no shapes where located within a tile the script would create an entry within a separate file called initTiles.json. This file is used at the initialization of the client application. It sets the loaded value of each tile stored in the initTiles file to true. This prevents unnecessary requests to the server for tiles that are definitely empty.

4.1.2 Publication Records



(a) Original record-gadm_id relation.



(b) Processed and simplified record-gadm_id relation.

Figure 4.5: Comparison between the original database structure and the preprocessed simplified structure to query publication records.

As previously stated, this application mainly focuses on the use of publication records of scientific papers and texts. All publication entries in the `record` table were gathered by the Centre for Social Innovation (ZSI) from the bibliographical databases Web of Science [WOS] and Scopus [SCO]. Scopus holds at the time of this thesis about 53 million entries, and Web of Science about 23 million entries. There exists an overlap of about 60% between both databases. ZSI has at the time of this writing about 3 million records from both databases unified into a single one. The records chosen by the ZSI mostly contain publications from Austria.

Using the relation to an author table and another relation to a table with institutions, each record can be mapped to a single longitude and latitude coordinate and henceforth to a GADM id. This relation can be seen in Figure 4.5a. These coordinates were already geolocated from the address of each institution from the people of the ZSI. In order to increase performance and reduce the server load on runtime, the necessary joins to join

the record table to the author table to the institution table with the correct GADM id, a preprocessed table named record_gadm was generated. The preprocessed joined table consists of entries with a record id and a GADM id (see Figure 4.5b). The only downside to using the preprocessed table is, that it has to be updated when new Records are inserted or new institutions have been geolocated. But that doesn't happen too often and the update can be done with a single SQL statement. Without the use of the preprocessed table the query would have taken too much time to compute the result. Even simple queries without any kind of filtering would have taken at least a second to execute. With the use of the preprocessed table the time needed to execute simple queries, without using any kind of filter, was reduced to an average of 50 ms.

Algorithm 4.1: Get values of second administration level without any filter.

```
SELECT
 COUNT(DISTINCT(rg.record_id)) as n,
 (gadm_ids[1] || '_' || gadm_ids[2]) as adm
FROM
 record_gadm rg
WHERE
 (gadm_ids[1] || '_' || gadm_ids[2]) IN ('243_15', '244_1')
GROUP BY gadm_ids[1], gadm_ids[2]
```

Algorithm 4.1 shows how a simple query of quantitative values looks like. The query returns tuples of quantitative values called n and correctly formatted (concatenated with underscores) GADM ids called adm.

Each record is given a list of GADM ids, in the stringed format described in the previous section, that are located within the viewport. Using the `WHERE` statement only records of visible gadm ids are being queried. This can be seen in the following example: `WHERE (gadm_ids[1] || '_' || gadm_ids[2]) IN ('243_15', '244_1')`. In this example a request for two administrative divisions on the second administration level is being requested. The gadm_ids array which hold the GADM ids on the record_gadm table, is being concatenated with underscores and brought into the correct format. After they have been formated, the PostgreSQL query selects all entries where the concatenated value is either 243_15 or 244_1. GADM ids are still stored on the server in the original array form, and must be concatenate dynamically on runtime to the currently necessary administration level. By storing them in the array form, they can be easily grouped by the `GROUP BY` statement.

Since it is possible to select the same record within an administrative division more than once, the `DISTINCT` statement has to be used. This issue could happen for example if the quantitative value of a country is determined but more than one institution within this country has worked on a paper. Using `DISTINCT` it is possible to only count records for the quantitative value, that are unique within an administrative division.

Algorithm 4.2: Adaption of query to return all GADM values.

```
SELECT
 i.adm as adm,
 COALESCE(rg.n, 0) as n
FROM (
  SELECT unnest( ('243_15', '244_1') ) adm
) as i
LEFT JOIN ( <rest of the query> ) as rg
ON i.adm = rg.adm;
```

Instead of returning the quantitative value zero if no records have been found within the GADM id, the PostgreSQL query would simply return no entry for this GADM id. But since the SeCho map always needs a value even if it is zero, the SQL query shown in Algorithm 4.1 had to be adapted to Algorithm 4.2. Using `unnest` another table is created on runtime with only GADM ids as entries. This table was `left joined` with the previously described query of Algorithm 4.1. This way all GADM ids present in the array that were unnested, would return a value. In the `SELECT` statement using the `COALESCE` function every value that doesn't have a value would be defaulted to return 0.

In order to get data where no records have been filtered the preprocessed table suffices by itself. To filter for bibliographic meta data a `JOIN` with the record table has to be performed. The actual filtering is done with additional `WHERE` conditions.

To filter for co-publications, the record_gadm table is joined with itself. As seen in the excerpt of algorithm 4.3, a table of record ids called `copub` is created. Using a `WHERE` statement tells the server to only return record ids that were created by institutions within the administrative division with the given `copubGADMid`. The `concatADM[copubLevel]` value is used like described above to concatenated the GADM array fields to the copubLevel. The only difference is that in this case the level of the copub determines how many administration levels for the concatenation are used. This ensures that the user can always change to another administration level while the selection is not influenced by the change. Using the `RIGHT JOIN` with the initial record_gadm table called rg and the `ON` statement, only entries are left where at least one co-publication was done with the selected `copubGADMid`.

4.2 Server - Client Protocol

Since the client application regularly requests data from the server, websockets [Fet11] are used for most transmission with the server. Websockets don't need to establish a new TCP connection for every transmission. The TCP connection is established once at the initialization of the websocket and continues to stay open until either side closes the connection. This reduces the amount of network traffic between server and client

Algorithm 4.3: SQL query to select co-publications.

```
...
record_gadm as rg
RIGHT JOIN (
  SELECT record_id
  FROM record_gadm
  WHERE """ + concatADM[copubLevel] + """ = %(copubGADMid)s
) as copub
ON rg.record_id = copub.record_id
...
```

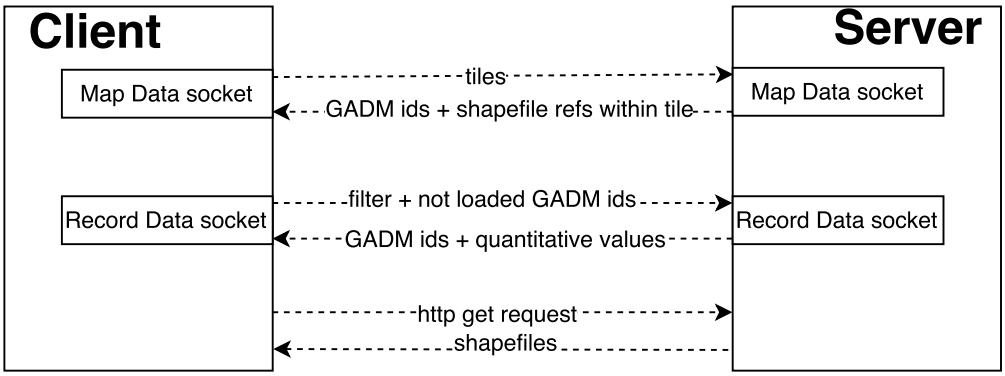


Figure 4.6: Simplified server - client protocol.

and increases the overall performance of each request. To prevent the need to parse what kind of request came from the client, two separate websockets were created. Both Websockets transmit data using JSON. Figure 4.6 shows all interactions between the server and the client at runtime.

4.2.1 Map Data Websocket

The Map Data websocket handles requests regarding tiles. The client sends the server a list of tiles and receives a list of GADM ids and shapefile references back.

The format needed for the request can be seen in Algorithm 4.4. The format for the response from the server can be seen in Algorithm 4.5. In the following, all attributes used in the request and the response object are being explained.

- The attribute `taskRef` is used in both protocols for references on the client application. The same integer value that was being sent by the request object will be transmitted with the response object. This was done so that the client application knows which response belongs to which request so that it can store and render the incoming data correctly.

Algorithm 4.4: Request format of the Map Data Socket.

```
{
  taskRef: int
  tiles: [{
   index: [int, int, int],
   rect: {left: int, right: int, top: int,bottom: int,level: int}
  }]
}
```

Algorithm 4.5: Response format of the Map Data Socket.

```
{
  taskRef: int
  tiles: [{
   index: [int, int, int],
   data: {geotiles: null || [string], contents: [string]}
  }]
}
```

- In both the request and response objects a list with `tiles` is transmitted.
- In both protocols the `index` is a three dimensional integer array that remains the same and tells the client to which maptile the response data belongs to.
- The request object contains a rectangular `bounding box`, which describes the extents of the given tile and its administration level.
- The response object contains a `contents` attribute, which holds a list of formatted GADM ids (concatenated with underscores).
- Additionally it also contains a `geotiles` attribute, which holds a list of shapefile references or a null value if no shapefiles have to be rendered in this tile.

4.2.2 Record Data Websocket

The Record Data websocket handles all request regarding publication data of each administrative division. At the start of this websocket the server sends data regarding the initialization of filter to the client application.

Other than the initialization, the client normally sends a request object like Algorithm 4.6 with level, filter and a list of ids to the server and receives a response like Algorithm 4.7 containing a list of record data back.

- As with the map data objects, both the request and response objects of the record data socket contain a `taskRef` attribute that is used by the client application to identify the response to a given request.

Algorithm 4.6: Request format of the Record Data Socket.

```
{
 taskRef: int,
 level: int,
 filter: {FILTER},
 ids: [string]
}
```

Algorithm 4.7: Response format of the Record Data Socket.

```
{
 taskRef:int,
 level:int,
 recData[{id:string, n: int}]
}
```

- Additionally both objects also hold an attribute for the administration `level` that this object refers to.
- The request object contains an object of possible `filter` options. If no filter was used, the object is empty. If a filter is used the object only contains attributes that are used for filtering with the conditions as values.
- The last attribute the request object contains is a list of GADM ids. The ids are in the underscore concatenated format.
- The response object contains a list with record data. These record data objects always have a reference to the GADM id and a value n for the amount of records that are located within the administrative division.

Possible filters and their formats include one or all of the following attributes:

- `sm:[string]` - a list of classifications (ids),
- `year:{start:int,end:int}` - an interval that describes when a record has been published,
- `timesCited:{start:int,end:int}` - an interval that describes how often a record has been cited,
- `authorCount:{start:int,end:int}` - an interval that describes how many authors have worked on this publication,
- `involvedCountries:{start:int,end:int}` - an interval that describes how many different countries have collaborated to create the publication,
- `id_scopus:boolean` - describes whether or not it is indexed in the Scopus database,

- `id_wos:boolean` - describes whether or not it is indexed in the Web of Science database,
- `citable:boolean` - describes whether or not the publication can be cited, and
- `copub:string` - only shows all co-publications with the given GADM id.

4.2.3 Shape Files

The shapefiles which hold the shapes of administrative division are transmitted with the normal TCP Protocol. These are just TopoJSON files located on the server and are, unlike the record or tile information, not calculated on runtime since they only change rarely. A shapefile only contains administrative division of one administration level. The transmission using the TCP Protocol has one significant advantage over websockets. The requested files can be cached by the clients browser automatically and therefore reduces the server and network load.

4.3 Server

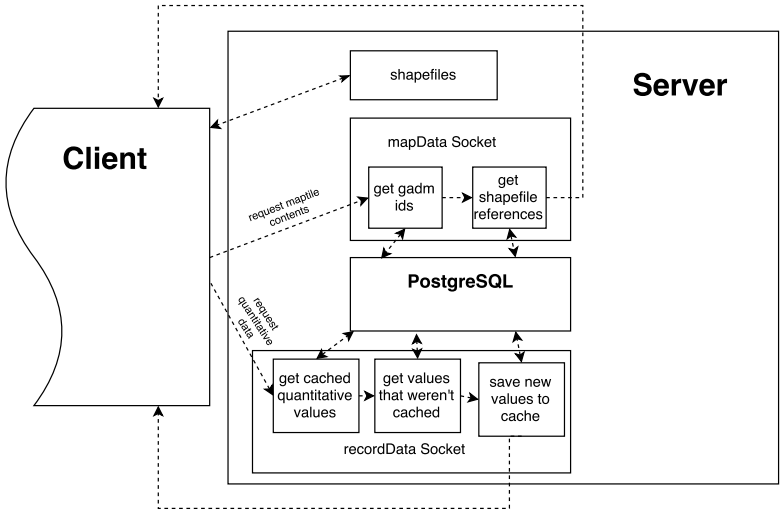


Figure 4.7: Diagram that shows how the server is structured.

The server was implemented using Python3 with the Tornado web framework [TOR] and Tornado's websocket handlers. To query data from the PostgreSQL database [POSb] the module `psychopg2` [PSY] was used. Since some more costly requests to the Postgres database were blocking all other requests to the server, the Python module `asyncio` [ASY] was used to allow the Python script to handle other requests while it waits for the result of an asynchronous database request. In order to make the `psychopg2` module asynchronous, the module `aiopg` [AIO] was used. Since the change to an asynchronous server meant that multiple database requests could be sent a database pool with a

minimum of 10 database connections and maximum of 20 connections was created. In Figure 4.7 the overall structure of the server can be seen.

4.3.1 Normal Behavior

When the server starts running, websockets are being initialized and the database connection pool is being created. Additionally the tornado web framework also provides the static HTML, CSS, and JS files needed to display the client application and the TopoJSON shapefiles. All static files provided will be gzip compressed by Tornado before transmission to reduces the network load.

Requests to any websocket are parsed automatically and checked on correctness. If any kind of error happens the current request is dropped and an error JSON object is returned to the client application. Otherwise it generates the necessary SQL statements depending on the requested data and binds the data at execution time as prepared statements to prevent the risk of SQL injections [TWX09].

4.3.2 Caching

Some requests to the PostgreSQL database are too costly. The cost of the query is influenced by how many entries are being selected from the database. Basic queries without any kind of filter return results in an appropriate amount of time (1 ms to worst cases of 500 ms) Depending on the kind of filter the query time can increase significantly. Filter that only remove few publications from the quantitative values, can increase the time to execute the query to 5 seconds or more. On the other hand, filter that remove many entries from the resulting quantitative values are quite efficient and can have execution times of less than 100 ms. To minimize the costs of all queries a caching method was implemented on the server. How the caching is being implemented on the database can be seen in Figure 4.8.

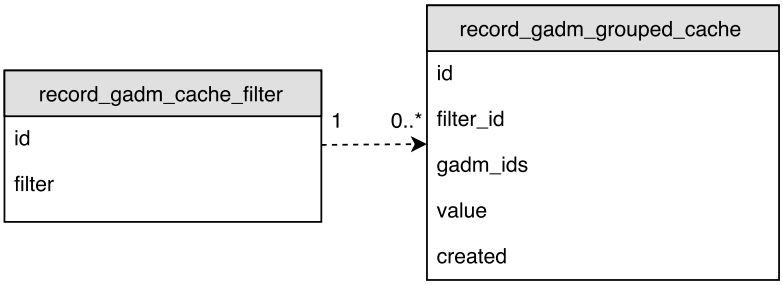


Figure 4.8: Tables that cache quantitative values.

On every record data request the server looks up the caching table, `record_gadm_cache_filter`, whether the requested filter was already used. If the filter was already used it returns an id with which another table called record_gadm_grouped_cache can be queried. In this table tuples of filter id, GADM id, created date and quantitative value exist. It returns

the cached quantitative values for entries with the given filter id and GADM id which aren't older than a certain time interval. If the GADM id with the filter isn't in the table or is older than the time interval of 100 days, the current values are calculated. After the new quantitative values were evaluated from the database they are upserted to the caching table. Upserting means that a new entry is inserted or an existing one is updated depending if an entry already exists in the table.

4.4 Client

The client application uses HTML, CSS and JavaScript to display the choropleth map to the user. To render shapes of administrative divisions the JavaScript library for data visualization named D3 [D3T] was used in conjunction with its TopoJSON interpreter extension. To enable asynchronous TCP connections with the server to load shapefiles from the server the `queue plugin` of D3 [D3Q] was used. A settings object was created to allow all functions or objects within the application to access variables stored in it. Variables in the settings object can be adjusted before the client application gets initialized. This can also be done within a separate JavaScript file to allow a separation of modules.

Everything is drawn on an SVG HTML element. Every administration level used on the SeCho map has their own SVG group element. All shapes of level 0 are located on a different group element as every shape of level 1 and so forth. Once a shape has been added to a group element, it will always stay there and never be removed. Each group layer is ordered so that a lower administration level (e.g. level 0) is drawn first, and higher administration levels, are drawn last and on top of less detailed levels. Every levels group element is located in an overall group element called map. This map group element is used to apply the zoom and the panning actions by using the CSS transforms scale and translation.

4.4.1 UI Manager

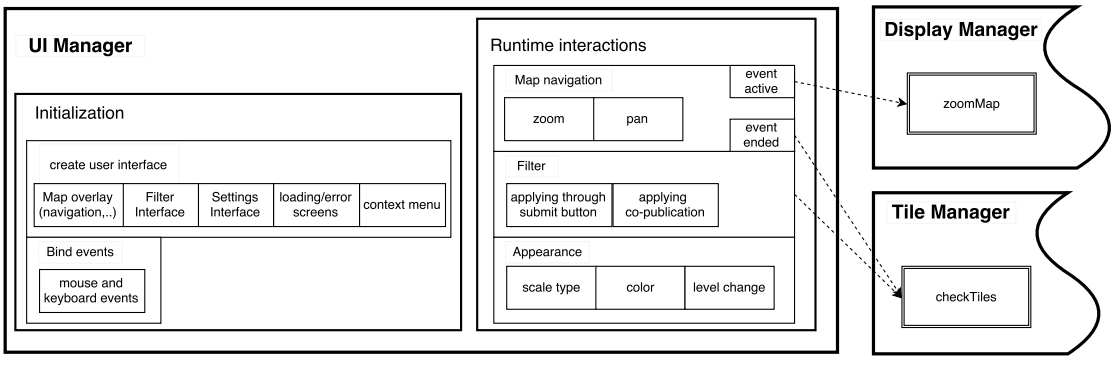


Figure 4.9: Pipeline of the UI Manager.

The `UI Manager` represents the layer between the user and the inner workings of the client application. As you can see in Figure 4.9, it handles the initialization of the user interface and interactions with the interface during runtime. The two main parts of the application the UI Manager interacts with are the `Display Manager` and the `Tile Manager` where the execution of the pipeline continues. The `UI Manager` first initializes the user interface for the navigation, loading and error screens, filter and settings windows, and maps the correct events to each element. At the initialization, the UI Manager also adds a context menu, which is used to provide more information about a selected tile. Additionally, this context menu also contains the button to select the administrative division for co-publications. Furthermore it handles basic interactions with the application and distributes the calculation results to more specific methods.

The zooming and panning of the map can be done in three different ways. All three methods use D3's zoom behavior to navigate the map. The zoom behavior is being applied to the HTML SVG element and maps all mouse interaction on this element directly to change the current zoom level or pan offset. These values can also be changed directly by JavaScript methods. The first and most obvious way to navigate the map uses the mouse. The user can drag and drop for panning or use the mouse wheel for zooming. By using the mouse, the zoom behavior can detect when a zooming/panning action starts, ends or is currently active. It is possible to map callbacks to these functions, which get called after the current zoom and pan values have been determined by the zoom behavior. During the active phase of the zoom/pan event the self implemented callback only pans and zooms the map using CSS transforms on an SVG group. This svg group holds all administrative divisions of the map. Only the callback after the zoom event ended starts the next section of the pipeline, which determines if new map tiles became visible during the navigation event and loads data if necessary. Another way to navigate the map can be done using navigational buttons on the map or by using keys on the keyboard. Both ways call functions to determine how much the map was panned or zoomed, write those new values to the zoom behavior and call the same callback used by the zoom end function.

All filter are being generated on initialization of the application. Their default values are being set according to the settings object. Filters are being applied during runtime with the click on a submit button. After the button was clicked, the `checkTiles` method of the `Tile Manager` gets called.

4.4.2 Tile Manager

The `Tile Manager` handles everything regarding the viewport and determines which administrative divisions are displayed to the user. The overall pipeline of this module can be seen in Figure 4.10. Tiles are being generated on the initialization of the application. They are stored as objects in a three dimensional array. The array was constructed so that the first dimension indicates the administration level, the second to indicate the longitude start value and the third to indicate the latitude start value. Each tile contains two important attributes. The first attribute named `contents` is used to store a list

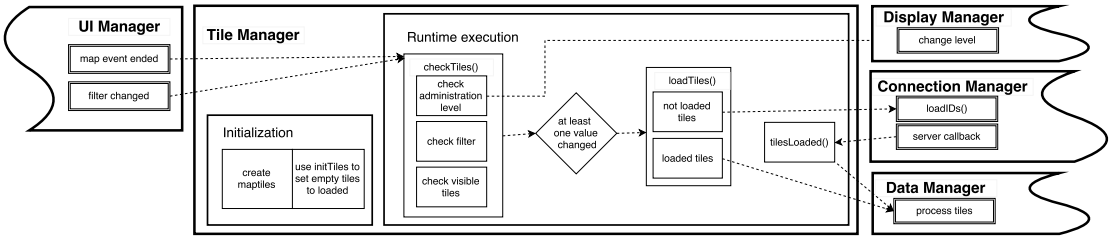


Figure 4.10: Pipeline of the Tile Manager.

of GADM ids. This list indicates which administrative divisions are at least partially within the tile. The second attribute named `geotiles` contains a list with references to shapetiles. This list is used to determine which shapetiles the application has to load to render everything within this maptile without any holes. Additionally, it also holds the bounding box of the current tile, an index that indicates the tiles location within this tiles array and a `loaded` boolean flag which tells the application whether or not the `contents` and `geotiles` of this map have already been loaded from the server. Soon after the initialization, the server provides the `initTiles` file. In this file a list of empty tiles exists. As soon as the server provided the data, the tiles array is traversed and every tile within this file is set to loaded.

On runtime, the first method that always gets called by the `UI Manager` is the `checkTiles` method. This method calls all other methods to determine the current state of the map. It checks whether or not the current filter was already used by the application. The `checkTiles` method also calls other methods that determine the current administration level and visible tiles. To determine whether or not the administration level changed, the value set by the zoom behavior is checked against multiple intervals set by an array of discrete values. If the zoom value falls into a different interval the zoom level has changed. The administration level is being calculated by determining in which interval the zoom value lies. After all necessary values have been determined, the `checkDisplay` method is being called. This method compares these values to the values of the previous state of the map. If any value has been changed the application pipeline continues with more costly methods which load and render data. After a change in the administration level has been determined, the `changeLevel` method of the `Display Manager` is being called.

After it is determined that the map changed from the previous state, it checks whether or not any tile data has to be loaded from the server. If there are tiles that haven't been loaded, a `Connection Manager` method to load the contents of the tile gets called. Information about the current state of the map, like other previously loaded tiles, current administration level and the filter, are saved within a `TaskQueue` object. This object returns a `taskRef`, an integer value which can be used later to load the correct state information and continue with the application. This is especially helpful if in the meantime the user has made another user input. This way data is still being loaded, stored and rendered correctly. The reference given by the `TaskQueue` has to be

transmitted to the server so that the response of the server can return the same value to the client. This way, the previous map state can be continued.

While the client application waits for tiles to be loaded, the client application continues the pipeline with all tiles that already have been loaded. This way the user receives faster feedback from the SeCho map and knows that his interaction with the map was is being processed.

After the server provided the data of the previously not loaded tiles, they are being stored in the correct tile of the tiles array and the `loaded` flag is set to true. Afterwards all tiles, whether they were newly loaded or previously loaded, are sent to the `Data Manager`.

4.4.3 Connection Manager

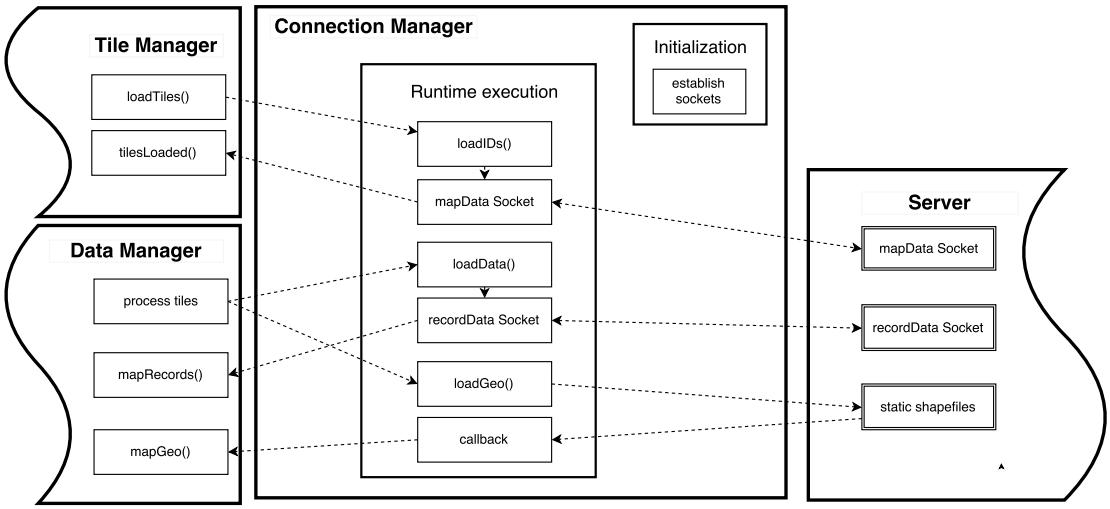


Figure 4.11: Pipeline of the Connection Manager.

The `Connection Manager` handles all communications between the Server and the client. The Figure 4.11 shows the inner workings of the `Connection Manager`. At initialization of this method, it creates both websocket connections using a self created socket method. At runtime it handles the formatting and sending of data objects from and to the server and their respective callbacks to the rest of the application. Additionally, it resends objects if something went wrong during transmission. Websockets are being restarted after a given time if the socket connection between client and server was closed.

4.4.4 Data Manager

The `Data Manager` handles the storing, retrieving and processing of shape and publication data. In Figure 4.12 the rough structure of this module can be seen. The first method of this object receives the task object used in the `Tile Manager` as an input argument.

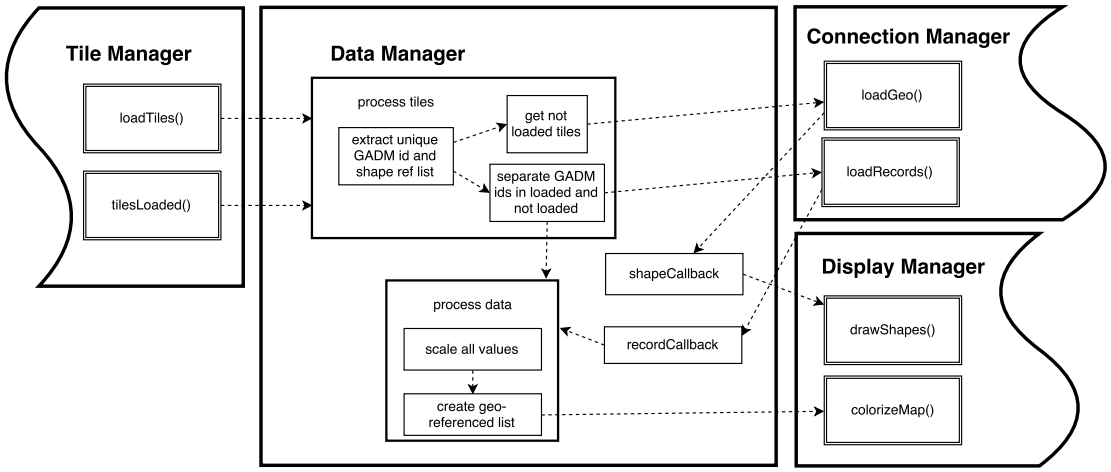


Figure 4.12: Pipeline of the Data Manager.

In this task object the visible tiles are stored amongst other things. From those tiles the `content` and the `geotile` attributes are being extracted and concatenated into two lists. From both lists items that occur multiple times are being removed so that every item only occurs once. After those lists have been created, the application determines which values it has to load from the server and which it has already stored in a cache.

To reduce server and network load, two levels of caching are being used within the `Data Manager`. The first level has a session scope and is stored in JavaScript objects. The session scope is being removed automatically by the browser when the site reloads or the tab closes. The second level which uses localstorage [Tri08] endures over multiple sessions and days.

The first level is stored within a map object from D3. For every administration level, a separate D3 map exists. The key for each map entrance is created from the GADM id and the `stored filter index`. The `stored filter index` is an integer that references the used filter. A list of all used filters is stored in the localstorage. From this list the current `stored filter index` can be determined by calculating the position of the current filter in this array. If the filter hasn't yet been added to the localstorage list, it is added. The value of the map contains the quantitative value of the GADM id with the given filter.

The keys and values used in the second caching level, the localstorage, are similar to the first caching level. Additionally, a `created` value is stored with the quantitative value. With this `created` value it can be determined whether or not the stored quantitative values are too old or not. Both values are stored in a JSON object in the localstorage.

To check whether or not a value has already been loaded the application first looks in the session cache, after it wasn't found the localstorage cache is checked. If the application found a value for the used filter and GADM id that isn't too old in the

localstorage cache, the data is also stored in the session cache. If the value for a certain administrative division wasn't found in any cache, the `Connection Manager` is used to load the correct quantitative values. Before the server is called, the current state data is stored again in a `TaskQueue`. The task object stored differs from the previously object. Instead of a list of visible tiles, it now stores the processed list of GADM ids and instead of the filter it stores the `stored filter index`. The `TaskQueue` returns again an integer reference value, which can be used later to restore the values of the current state. The `Connection Manager` is used again to send not loaded data to the server for processing. The `Connection Manager` calls the callback provided at the initialization of the socket as soon as the data has been processed by the server. The callback then processes the response, stores the incoming data in both caches and calls the next method to process all data for rendering. While the SeCho map waits for the server to finish loading the data, it already processed and rendered data that have been loaded previously.

In the processing method all loaded values that have to be displayed are gathered. If the scale is set to `automatically calculate` its values every time this pipeline is called, it is updated on this step. Afterwards a list of geo-referenced quantitative values is being created. All quantitative values that are in this list are scaled from its original value to a value between 0 and 9. After the processing has finished the `colorizeMap` method of the `Display Manager` is called.

While the quantitative data is being retrieved by the `Data Manager`, it also fetches the shapefiles where the polygons of the administrative divisions are being stored. Similar to the session caching of publication data, a map is created that indicates whether or not a certain shape file has been loaded. If a shape has been loaded once no further steps are needed since they don't have to be redrawn. If a shape hasn't been loaded yet, the `Connection Manager` loads each file and calls a callback on the `Data Manager` to store the data and propagate it to the `drawShapes` method of the `Display Manager`.

4.4.5 Display Manager

The `Display Manager` handles how the data is displayed to the user. Figure 4.13 shows how the `Display Manager` module is structured.

The `changeLevel` method, which is called from the `Tile Manager` when the administration level changes, hides more detailed layers if zoomed out. When the user zooms into the map, lower detailed administration levels are painted over by more detailed administration levels. This has the advantage that nearly every time something is being displayed to the user and he can still navigate the map effectively, while the rest of the data loads in the background.

The `drawShapes` method renders the shapes of administrative divisions on the layer. If for some reason another group element of another administration level is being displayed on the map, the administrative divisions are still added to the correct element. The method first processes the TopoJSON object, which holds the administrative divisions.

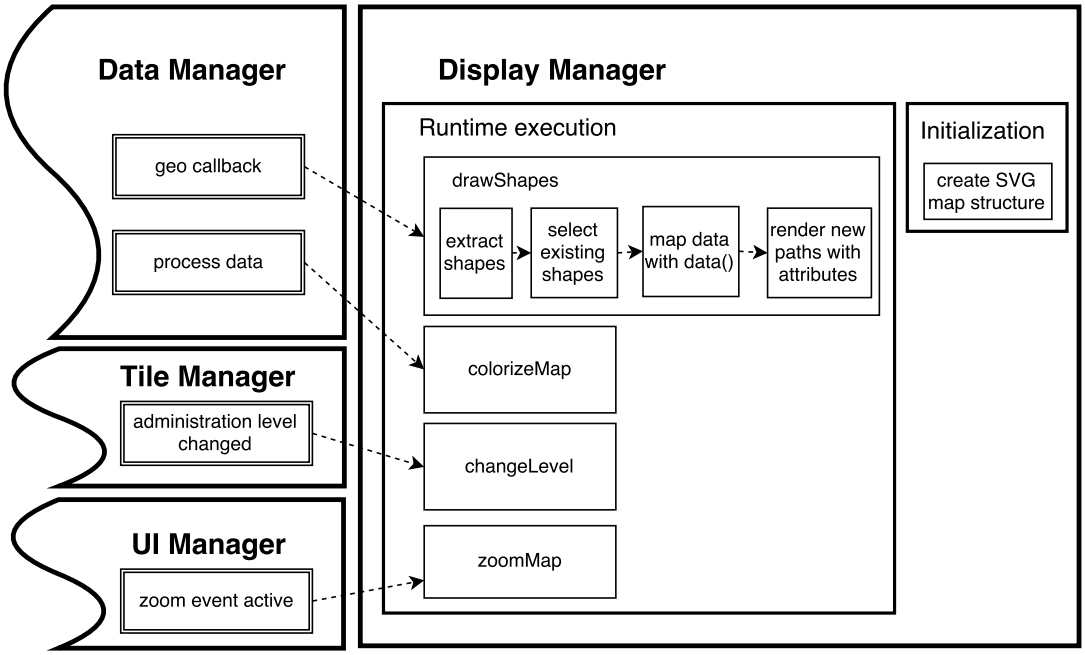


Figure 4.13: Pipeline of the Display Manager.

It uses D3's `topojson.feature()` method to extract all shapes and attributes from the TopoJSON object. Next, D3's `select()` and `selectAll()` methods where used to select the correct group element and every administrative division already rendered on this layer. Using D3's `data()` method, it is determined which shapes already have been added to the layer and which it has to add using the `enter()` method of D3. The `data()` method needs as argument the extracted TopoJSON features and additionally a second argument to map the data correctly. For this argument, the underscored concatenated GADM id, which was stored in the TopoJSON file in a preprocessing step, is used. If this argument isn't given to the method, D3 would think that some administrative divisions were already rendered on the map. This would cause that these shapes will not be added to the map using the `enter()` method. Using D3's `enter()` method, the administrative divisions are being appended as SVG path elements with some meta data to the layer. It is important that every path has its own CSS id, which is essentially the concatenated GADM id prepended with a letter to be backwards compatible with older CSS id standards [CSS]. Additionally, every path also gets the `vector-effect non-scaling-stroke` [NON] and is bound to a click event to allow the use of context menus. `Vector-effects` [VEC] can be used for SVG elements to change the way they are drawn. The `vector-effect` used is not supported by all major browsers, but would allow that no stroke is being scaled, when the `CSS transform scale` is being applied to the element. If any browser doesn't support this attribute, a new stroke width is calculated when the zoom event ended.

The method `colorizeMap` renders the colors of each path element. Since it is pos-

sible that the colorization of the map is called before the `drawShapes` method, D3's `select()` method can't be used to colorize each shape since they don't exist yet on the layer. In order to avoid this, the color information is not stored on the path element but on a style sheet. Using JavaScript's method `deleteRule`, all rules in the style sheet are removed at the start of the `colorizeMap` method. This prevents the style sheet from getting too large, since otherwise it would add rules every time the method is called. Afterwards, for each GADM id, a rule to colorize the path is added to the style sheet using `addRule`. The color of each scaled quantitative value is determined using an ordinal scale provided by D3. This scale maps values from 0 to 9 to a string value which represents the appropriate color as a CSS rgb value (e.g. `rgb(35, 139, 69)`).

Since some countries only have one or two administration levels, a solution has to be found to visualize those administrative divisions. If this was not dealt with, there would exist a hole in the visualization, where the lower level administrative division would not be painted over by a higher administrative division since they do not exist. The PostgreSQL database stores GADM ids in the `record_gadm` table using arrays. Therefore we would still receive a valid GADM id with correct quantitative values from the server. The GADM id of those values would have one or multiple `_0` added to the GADM id where there is no administrative divisions. This property can be used to our advantage. The `colorizeMap` method removes all occurrences of `_0` in all GADM ids. This would result in the colorization of the lower level administrative division that is currently looking through the hole.

CHAPTER 5

Result

The implemented SeCho map can be accessed with any latest major browser. Differences between all major browsers have been identified and solutions have been found to guarantee that the SeCho map works and looks the same way with those browsers. The only major problem any tested browser had was a performance issue with Firefox's SVG CSS transforms. This is a known bug in Firefox that hasn't been solved for some years [FIR]. This problem was partly solved by simplifying shapes of the administrative divisions to a bare minimum. Nonetheless, there is still a visible performance difference between zooming and panning actions of Firefox and any other major browser. Firefox renders zooming and panning actions with an average of 3 FPS. Other browsers like Chrome or Opera render map interactions with an average of 100 to 20 FPS. If the administration level changes and more data has to be rendered the performance of Chrome and Opera reduces to about 7 FPS for a short amount of time.

The requirements given by the Centre for Social Innovation (ZSI) have all been addressed within this project. With the use of this SeCho map the people of the ZSI now can generate dynamic choropleth maps within a web application for the whole world and in three different administration levels. Previously, this was only possible for them with static choropleth maps. In addition to this, it is now also possible to view co-publications between administrative divsions. This allows them to see scientific relations between geographic regions. In addition, the people of the ZSI disclosed that the SeCho map may be used in a EU H2020 project, called European map of knowledge production, that will begin in January 2017. There users, which consist mostly of policy makers and analysts, can use this map to visualize which topics are researched where in Europe.

5.1 Performance

In Table 5.1 you can see how different scenarios effect the execution time of the application. Please note, that the execution times without any filter (Scenario 1) could'nt be assessed

administration level	uncached	server cached	client cached (localstorage)	client cached (session)
Scenario 1: Without any filter *				
level 0	3200 ms	3100 ms	2530 ms	5 ms
level 1	1050 ms	900 ms	450 ms	10 ms
level 2	1650 ms	1600 ms	1100 ms	25 ms
Scenario 2: Co-publications with Austria				
level 0	400 ms	150 ms	10 ms	5 ms
level 1	600 ms	300 ms	20 ms	15 ms
level 2	1450 ms	800 ms	40 ms	25 ms
Scenario 3: Co-publications with Italy				
level 0	150 ms	100 ms	10 ms	5 ms
level 1	250 ms	200 ms	20 ms	15 ms
level 2	650 ms	600 ms	50 ms	25 ms
Scenario 4: Filter: Show all but one classification				
level 0	2600 ms	50 ms	10 ms	5 ms
level 1	5000 ms	250 ms	15 ms	15 ms
level 2	5000 ms	700 ms	40 ms	25 ms
Scenario 5: Filter: Show publications between 2006 and 2012				
level 0	1300 ms	150 ms	10 ms	5 ms
level 1	1600 ms	300 ms	20 ms	15 ms
level 2	2100 ms	800 ms	40 ms	25 ms

Table 5.1: Average execution times of different scenarios from detection of user input to finish loading. Note *: Scenario 1 cannot be compared effectively to other execution times since it also loads and renders other data.

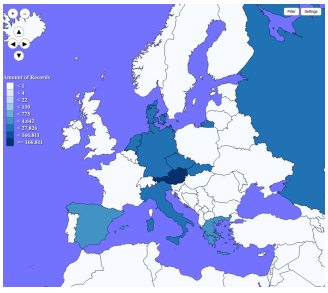
correctly. This is due to the initialization of the application and the loading of other application specific data like the contents of maptiles or the loading and rendering of shapetiles. The execution times shown in the table start after the detection of user inputs and end with the rendering of data. Apart from the first scenario, the other scenarios had shapefiles and maptile contents already loaded. If they weren't loaded beforehand the execution time would have increased by about 500 ms on average. As you can see in Table 5.1, the uncached scenarios have a wide range of execution times. Scenario 4 represents a filter that normally will not be used by the user. The database query only removes small amounts of data which increases its execution time significantly. Scenario 5 represents a normal filter query. A good amount of data is removed from the database selection which makes for a faster execution time. Scenario 2 represents the worst case for a co-publication selection because in the used dataset, most publications records belong to institutions within Austria. Scenario 3 represents a good example for the execution time of co-publications. Italy has many records but not as many as Austria. Therefore the execution time is significantly lower.

You can see that, apart from the first scenario and all uncached results, all cached execution times, with the same administration and cache level, are about identical. This is because every scenario has to do the same to retrieve those values regardless of used filter. The execution time of higher administration levels increases because more administrative divisions are visible. The overall execution times for the server cache might not seem too efficient for most queries, but if the used dataset increases in size, the serverside cache gets more relevant. To test this the used dataset was increased tenfold, and scenario 2 was tested. While the execution times of uncached values increased significantly to database query times of 20 seconds and more, the time stayed the same for cached values. This means that costly executions only have to be calculated once by one user and every other user would get the calculated data in a timely fashion.

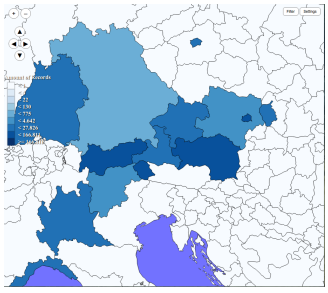
Using the session cache of the client, the quantitative data can be retrieved with the best performance. This performance is necessary in order to smoothly render the transition between administration levels. While the user navigates between areas he retrieved earlier, it is almost not noticeable that colors are re-rendered on runtime.

5.2 Use cases

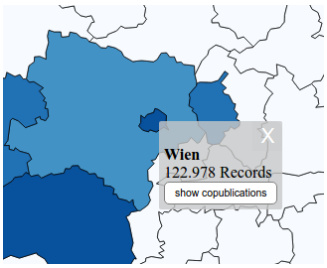
In the following sections two specific use cases, which are solved by the SeCho map, are described.



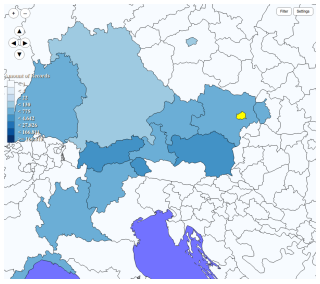
(a) Map at application start.



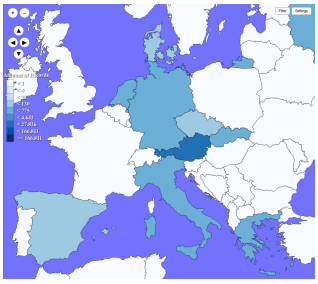
(b) Zoom to administration level 1.



(c) Context menu of Vienna.



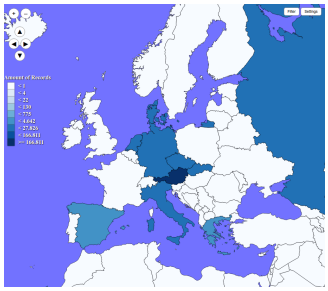
(d) Co-publications with Vienna on level 1.



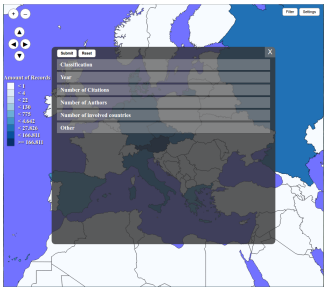
(e) Co-publications with Vienna on level 0.

Figure 5.1: Use case 1: Co-publications with Vienna.

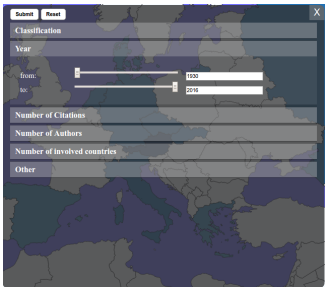
In the first use case the user wants to view all administrative divisions that have collaborated with Vienna. To do this the application is first accessed with the browser and the normal choropleth map of Europe is visible as seen in Figure 5.1a. Next the user zooms into the map, roughly targeting Vienna with his cursor, so that the second administration level is visible (seen in Figure 5.1b). The user than clicks with the mouse on Vienna and a context menu like Figure 5.1c becomes visible. On this context menu the user clicks on the `show co-publications` button. After that the query is send to the server and the quantitative data for each shape is retrieved. Now the map is re-rendered with Vienna having a distinct yellow color. This can be seen in Figure 5.1d. In the last step the user zooms out again to administration level 0. He can now see all countries that have collaborated on papers with Vienna as seen in Figure 5.1e.



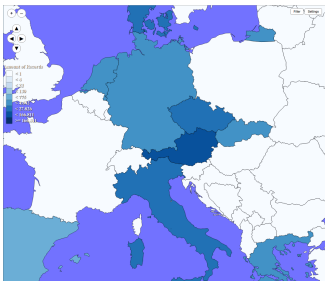
(a) Map at application start.



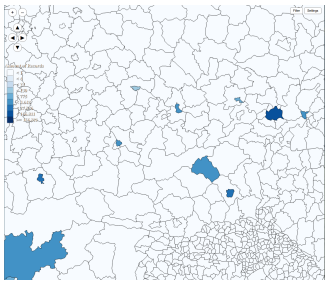
(b) Filter menu.



(c) Filter menu for year.



(d) Year filtered on administration level 0.



(e) Year filtered on administration level 2.

Figure 5.2: Use case 2: Filter the year.

In the second use case the user wants to display only publications that were published between 2010 and 2012. To do this he first loads the browser application and displays the unfiltered map as seen in Figure 5.2a. He then clicks on the filter button on the upper right corner to display the filter menu seen in Figure 5.2b. Here the user opens the correct filter option by clicking on `year` so that the values of this filter are visible (seen in Figure 5.2c). He now either slides the slider to the desired value or writes the desired start and end year in the adjacent white boxes. Now he only needs to click on the `submit` button to apply the filter. The user closes the filter menu with a click on the `X-button` on the upper right corner of the menu. After the client retrieved and rendered the new data, he can see the filtered SeCho map (see Figure 5.2d). Now the

user zooms into the map to display the second administration level as shown by Figure 5.2e.

CHAPTER 6

Conclusion and Future Work

With the use of semantic zooming, the SeCho map introduces a feasible approach to visualize multiple administrative levels on one map. Users can first view the map as a whole and later choose to view more details of regions they are interested in. By visualizing the SeCho map in a web browser, everyone with a computer and Internet connection, regardless of operating system, can access information visualized by the map without the need to install an application. Furthermore the data that is being used can be stored on a server.

Due to the settings object and a separation between server and client, it is easily possible to change the source of the data set. To do this, only the record socket link has to be changed to the correct socket and a new socket on the server has to be created that handles the new data set. Depending of the type of data set, the filters might also have to be adapted.

Another future work that could be implemented for the SeCho map is the use of multivariate data. To to this, a second map with another data attribute visualized might be introduced. Both maps could either be rendered side by side or on top of each with the possibility to jump back and forth between both maps. In both cases, the pan and zoom level of both maps is always the same to allow an easier comparison between both maps. Else it might also be possible to display multivariate data using other visualization techniques, like a bar chart, when showing the context menu of a selected administrative division.

A third possibility to enhance the SeCho map can be implemented introducing another last and most detailed level of the semantic zoom. This means after the most detailed administration level of the choropleth map was reached, the map can be zoomed in even further to reveal another kind of visualization. On this level of semantic zooming, the choropleth map might change into an isopleth or proportional symbols map and visualize each research institution alone without being grouped by the administrative division.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AIO] aiopg. https://aiopg.readthedocs.io/en/stable/. Accessed: 2016-09-01.

[ASY] Asyncio. https://docs.python.org/3/library/asyncio.html. Accessed: 2016-09-01.

[CHVKS10] Sergio Cabello, Herman Haverkort, Marc Van Kreveld, and Bettina Speckmann. Algorithmic aspects of proportional symbol maps. *Algorithmica*, 58(3):543–565, 2010.

[COL] Colorbrewer 2.0. http://colorbrewer2.org. Accessed: 2016-09-01.

[CSS] CSS ID standard. `https://www.w3.org/TR/html-markup/global-attributes.html#common.attrs.id`. Accessed: 2016-08-29.

[D3Q] D3's Queue Plugin. https://github.com/d3/d3-queue. Accessed: 2016-09-01.

[D3T] D3. https://d3js.org/. Accessed: 2016-09-01.

[DBBCM04] Mark De Berg, Prosenjit Bose, Otfried Cheong, and Pat Morin. On simplifying dot maps. *Computational Geometry*, 27(1):43–62, 2004.

[FBC00] A Stewart Fotheringham, Chris Brunsdon, and Martin Charlton. *Quantitative geography: perspectives on spatial data analysis*. Sage, 2000.

[Fet11] Ian Fette. The websocket protocol. 2011.

[FIR] Firefox performance problem. `https://bugzilla.mozilla.org/show_bug.cgi?id=820272`. Accessed: 2016-08-30.

[GAD] Global Administrative Areas. http://gadm.org/. Accessed: 2016-09-01.

[GEO] GeoJSON. http://geojson.org/. Accessed: 2016-09-01.

[GMA] Google Maps. https://www.google.com/maps. Accessed: 2016-09-01.

[MAPa] MapShaper. https://github.com/mbloch/mapshaper. Accessed: 2016-09-01.

[MAPb] Mapshaper Visvalingam Simplification. https://github.com/mbloch/mapshaper/wiki/Command-Reference#-simplify. Accessed: 2016-08-28.

[MMM14] Lukas M Marti, Robert Mayor, and Shannon M Ma. Tiling of map data, September 30 2014. US Patent 8,849,308.

[NON] Vector-Effect: non-scaling stroke. https://www.w3.org/TR/SVGTiny12/painting.html#NonScalingStroke. Accessed: 2016-08-29.

[OK10] Ferjan Ormeling and Menno-Jan Kraak. *Cartography: Visualization of Spatial Data*. Guilford Press, 2010.

[OSM] OpenStreetMap. https://www.openstreetmap.org/. Accessed: 2016-09-01.

[PF93] Ken Perlin and David Fox. Pad: an alternative approach to the computer interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 57–64. ACM, 1993.

[POSa] PostGIS. http://postgis.net/. Accessed: 2016-09-01.

[POSb] PostgreSQL. https://www.postgresql.org/. Accessed: 2016-09-01.

[PSY] Psycopg2. http://initd.org/psycopg/. Accessed: 2016-09-01.

[Rai34] Erwin Raisz. The rectangular statistical cartogram. *Geographical Review*, 24(2):292–296, 1934.

[SCO] Scopus. https://www.scopus.com/. Accessed: 2016-09-01.

[SMKH09] Terry A Slocum, Robert Bach McMaster, Fritz C Kessler, and Hugh H Howard. Thematic cartography and geovisualization. 2009.

[TOP] TopoJSON. https://github.com/mbostock/topojson. Accessed: 2016-08-28.

[TOR] Tornado web framework. http://www.tornadoweb.org. Accessed: 2016-09-01.

[Tri08] Alberto Trivero. Abusing html 5 structured client-side storage. *EB/OL]. http://packetstorm. orionhosting. co. uk/papers/general/html5whitepaper. pdf*, 7, 2008.

[TWX09] Stephen Thomas, Laurie Williams, and Tao Xie. On automated prepared statement generation to remove sql injection vulnerabilities. *Information and Software Technology*, 51(3):589–598, 2009.

[V+10] Kevin Verbeek et al. Necklace maps. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):881–889, 2010.

[VEC] Vector-Effects. https://dev.w3.org/SVG/modules/vectoreffects/master/SVGVectorEffectsPrimer.h Accessed: 2016-09-01.

[VW13] Maheswari Visvalingam and JD Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, 2013.

[WOS] Web of Science. http://webofknowledge.com. Accessed: 2016-09-01.

[ZSI] Centre for Social Innovation (ZSI). `https://www.zsi.at/`. Accessed: 2016-09-01.