Department of Informatics
Technische Universität München
Chair for Applied Software Engineering

MASTER'S THESIS IN INFORMATICS

DEVELOPMENT OF A VISUALIZATION SYSTEM FOR THE INTERACTIVE EXPLORATION OF LINKED DATA

BY ALEXANDER WALDMANN

Department of Informatics

Department of Informatics
Technische Universität München
Chair for Applied Software Engineering
In cooperation with: MIT AgeLab

MASTER'S THESIS IN INFORMATICS

DEVELOPMENT OF A VISUALIZATION SYSTEM FOR THE INTERACTIVE EXPLORATION OF LINKED DATA

Explore complex data with appropriate visual tools

ENTWICKLUNG EINES VISUALISIERUNGSSYSTEMS ZUR INTERAKTIVEN UNTERSUCHUNG VON VERLINKTEN DATEN

Komplexe Daten mit angemessenen visuellen Werkzeugen explorieren

AUTHOR: ALEXANDER WALDMANN
SUPERVISOR: PROF. BERND BRUEGGE, PH.D.
ADVISOR: BRYAN REIMER, PH.D.
DATE: 07/15/2014

ERKLÄRUNG

Ich versichere, dass ich diese Master's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I confirm that this master's thesis is my own work and I have documented all sources and material used.

München, den 10. Juli 2014, Alexander Waldmann

Creativity needs freedom.

— Mantra of the University of Heidelberg

ACKNOWLEDGMENTS

This work is dedicated to my family and my girlfriend. It is to my sister who strengthened me in my belief that science must be pursued with much endurance. It is to my father who teached me that it requires no less than the curiosity of a child. It is to my mother who convinced me to take a step back when needed. And it is to my girlfriend who shared the beauty and pain of this academic journey.

I thank all my collegues, especially my supervisor Bryan Reimer for the opportunity to work with him and experience the dense field of the sciences pursued at the MIT AgeLab and Adrian Rumpold for his technical expertise. I owe a debt of gratitude to Jonathan Dobres for proof-reading this thesis and his suggestions. Finally, I thank my advisor Prof. Brend Brügge for his support and the opportunity to write my thesis at his chair.

ABSTRACT

In the age of big data, very large sets of different measurable data are permanently acquired. However, the development of efficient methodology to search for and identify the actual information hidden in large data sets, and to explore relationships between subdatasets, is lagging behing. Data visualization is an efficient and intuitively accessible approach to identify patterns in large and diverse data sets. State-of-the-art computing provides the enabling technology to visualize data not only as static images but also in interactive visualizations. But even the most efficient and powerful data visualization tools do not satisfactorily address and exploit the possibilities of interactive data analysis. They constrain creativity and are often complex to use. In this work I propose to use "component based visualization" for interactive data exploration. Component based visualization derives properties such as "composability" and "information hiding" from the concept of a software component and transfers them to the domain of data visualization. By building a directed graph of such components, a complex set of visualizations can be composed. The dataset of a component itself can then be used as a parameter to define the semantics of an arc between two components (e.g. two visualizations can be connected to show the same GPS coordinate from different data recordings, thus allowing the comparison of other associated data, such as a heart rate). Additionally, users are allowed to not only interact with the visualization but also alter data processing and visualization by a component. This approach allows simulataneous exploration of multiple datasets applying various criteria, while comprising the relationships between the data. This concept is showcased through a visualization system that allows the user not only to dive into data via visualizations, but facilitates the creative freedom to build, extend and define new visualizations and their composition. The prototype, called DAVID (dynamic analysis and visualization of integrated data), is a tool for interactive exploration of linked data.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

Part I

INTRODUCTION

1 PROBLEM STATEMENT

The digital revolution, the "change from analog mechanical and electronic technology to digital technology" [79] has led into a new age in which "an economy based on information computerization"[79] has evolved. In this age, computation and information itself, in all its different incarnations, play a central role in both business and science. Information storage grew from 54.5 exabyte in the year 2000 to 295 exabyte in 2007 [82] (world wide). Today, the amount of data available on the Internet is estimated to grow by 4 exabyte each day [67]. Owing to the increasing affordability of memory storage and high-power processors in the last decade [48], it is now possible for many research fields to collect massive amounts of data. In [46], the authors state that "researchers from the University of Berkeley estimate that, every year, about 1 Exabyte of data are generated, of which a large portion is available in digital form".

Additionally, the "smart home" and the "internet of things" is appearing on the horizon. In the last 10 years the prices for sensors have dropped significantly as well. With devices such as Google's "Glass"[1] or "Nest"[2] and Apple's "iPhone"[3], information is gathered all around us at any time. "Ubiquitous computing", the idea that "computing is made to appear everywhere and anywhere"[74], is not the distant future anymore but has become reality.

This development comes with unique opportunities and new challenges. Data analysis is often a highly complex task that cannot be streamlined, and requires significant domain knowledge as well as technical expertise. Numerous academic fields, such as Bio-Informatics and Geo-Informatics, have arisen out of the opportunities of "big data" even as its challenges put pressure on more traditional fields. Though the sheer amount of data is a challenge in itself, not only the size of the data-sets are problematic. The structure of the data is also of high importance. Data are often linked with each other in different ways, for example by correlation of values or similar semantics. Knowing and understanding those links is crucial to efficient and meaningful information extraction. In his book "The age of context", Robert Scoble summarizes that "it's not the big data mountain that matters, it's those tiny little spoonfuls we extract whenever we search. [...] [They] enable us to keep up with, and make sense of, an accelerating world" [67].

One common way to analyze such complex data, especially data that cannot be interpreted in its original encoding, is to find or develop viable visualizations to show "information that has been abstracted in some schematic form, including attributes or variables for the units of information" [35], thus allowing researchers and information consumers to "gain insight into an information space"[26]. Visualizing data allows one to access the information hidden in the data. As visualizations abstract, reduce, and compress data, it is crucal to choose a valuable representation in the context of a specific use case. Thus, data analysts need powerful new tools to assist them in the task of visualizing data and the information hidden in it. Paradoxically, visualizations now often begin

1 Project website: http://www.google.de/glass/start/what-it-does/
2 Product website: https://nest.com/
3 Product website: https://www.apple.com/iphone/

without a specificed target, since potentially relevant patterns in the data remain hidden prior to visualization. Consequently, data visualization tools need to support the data analyst in the creation and *exploration* of data, while also providing a means of creating the final representation.

In the past decade numerous tools, frameworks and even languages have arisen that focus on this task in one way or another. Statistical languages such as R support data exploration on a programmatic level, whereas libraries such as D3 allow for manipulation of the visualized elements. None of them, however, allow for an interactive exploration of data while maintaining the degree of freedom a data analyst needs. Most systems either focus on one specific aspect of data visualization, thus increasing the complexity of the system and the user's interactions with it, or the system may serve a wide variety of use cases, resulting in a more generic tool that is incapable of producing complex visualizations without substantial manual effort.

Here I propose a system that combines these aspects into one system by integrating many approaches and libraries developed over the last decade.

1.1 APPROACH

In this work I investigate the current state of interactive data visualization and its weaknesses when used to explore and interpret complex datasets. I present a system that allows for dynamic explorations of data while maintaining the creative freedom needed by data analysts. I focus on best practices not only in the field of data visualization but also in the area of Software Engineering. The goal of this thesis is to showcase a system that is capable of visualizing complex data in an interactive way. Such a system shall provide ample opportunities for refinement and customization by data analysts.

This work can be divided into four areas of investigation:

1. Visualization: I explain and discuss the fundamentals of the still young field of data visualization. In Chapters 3 and 4 I present typical workflows and visualization tasks commonly encountered by data analysts, and I argue for the necessity of creative freedom in this area.

2. Interactivity: As part of the area of "Visualization" (4), I investigate what "interactivity" in the context of data visualization means and what makes it valuable to analysts, designers, scientists, and data explorers who search for informative patterns in their data.

3. Constraints: In Chapter 2 the conflicts between these areas are discussed before describing each in more detail. In this chapter I point out that even though a vast set of powerful visualization tools, mechanisms, and frameworks exists, one must be chosen for practical use, depending on the use case. Once a system is picked, it becomes difficult or impossible for analysts to incorporate it into another system (for example if the use case changes). The system described in 5 and implemented in 6 acknowledges this problem and is based on the idea that the incorporation of many frameworks and libraries for data visualization is a necessity. This supports not only the freedom of choice, but most importantly, the reuse of existing visualizations, code and workflows. The system also addresses the issue of usability

in complex visualization systems. In Chapter 2, I describe the problems with the "state of the art" systems and their usability.

4. Exploration: Data exploration is one of the main concerns of this work. In Chapter 5 I propose "component based visualization", which gives data analysts and designers the opportunity to define and extend the creation and behavior of visualizations without requiring deep technical expertise. This chapter builds upon the previous ones and introduces ideas on incorporating "big data" and interactivity into such a system. However, exploration also means giving the data analyst complete control over how a visualization works and interacts with the user.

1.2 SCOPE OF THIS WORK

This work is written in the context of ongoing projects at the Massachusetts Institute of Technology (MIT) AgeLab. The example data used stems from real scientific projects conducted between 2007 and 2014, and inspiration was drawn from undocumented programs, workflows already present in this laboratory and brain storming with potential users of the system. The context and the specific requirements of this system are described in detail in Chapter 2. This work aims to outline a prototype rather than a fully functional and "ready to sell" product.

Even though this work proposes a system for data visualization, it does not describe methods of drawing and animating data in the elemental sense. In many ways it stands on the shoulders of the developments in data visualization of the last decade. Still a young field, a variety of libraries have been written to abstract the complexity of drawing and animating elements on a screen (see Chapter 3). The mechanisms offered by integrated and well supported libraries are used and referenced heavily. Chapter 5.2.2 describes in detail the reasons for this decision. A description of the developed prototype is given in Chapter 7.1 and depicts many of the concepts described throughout this work.

1.3 CONVENTIONS OF THIS DOCUMENT

This document uses the following conventions to present information.

- All citations will be marked with two square brackets, linking the citation in the index with a unique key ([key]).
- An explanation of a word will be marked by a footnote.
- An abbreviation will be written out when it appears for the first time and will be added to the nomenclature at the end of this document.
- To present software models, the "Unified Modeling Language" (UML), version 2.4 is used. If the semantics of the UML are not required, a mix of undefined semantics and representations derived from the UML are used and explained for each representation.
- The inconsistently defined words data, information, and knowledge are used as defined in the following table . This definition follows the understanding of data,

Data	Computerized representations of models and attributes of real or simulated entities
Information	Data that represents the results of a computational process, such as statistical analysis, for assigning meanings to the data, or the transcriptions of some meanings assigned by human beings
Knowledge	Data that represents the results of a computer-simulated cognitive process, such as perception, learning, association, and reasoning, or the transcriptions of some knowledge acquired by human beings

Table 1: Definiton of data, information, and knowledge[26]

information, knowledge and wisdom as a hierachy as proposed with the "DIKW-Pyramid"[66].

2 CONTEXT

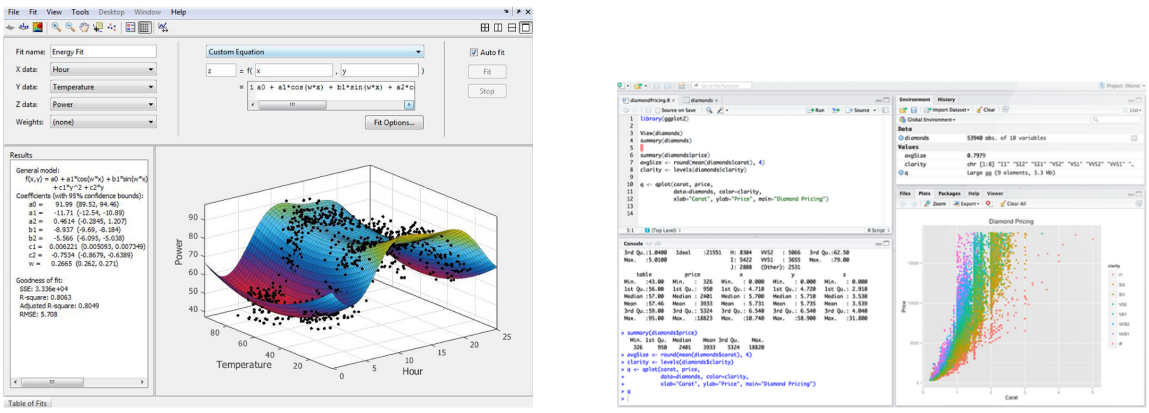
The visualization and manipulation of data are relevant to all scientific fields. As such, the topic of "data visualization" is one without clear boundaries, and can cover:

- Information visualization
- Interaction techniques and architectures
- Modeling techniques
- Multiresolution methods
- Visualization algorithms and techniques
- Volume visualization

[78] to give just a few examples. The first major challenge when investigating this rather broad field is to define a clear domain of interest. In this work I focus on the aspects of interaction techniques and architectures, and build upon the results of other works in the area of information visualization. Chapter 3 showcases several systems that are also relevant to these fields. To get a better understanding on interaction techniques it is helpful to look at state-of-the-art systems.

A selection of tools typically used for data analysis and visualization is shown in figures 1 and 2. The first figure shows two tools known for their strength in computation and analysis: MATLAB and RStudio. MATLAB is a "high-level language and interactive environment for numerical computation, visualization, and programming. Using MATLAB[1], one can analyze data, develop algorithms, and create models and applications"[54]. MATLAB grew out of the discipline of engineering, and prioritizes the handling of n-dimensional data matrices. It emphasizes computational efficiency but lags behind in

1 Product website: http://www.mathworks.com/products/matlab/index.html



Fit name: Energy Fit
X data: Hour
Y data: Temperature
Z data: Power
Weights: (none)

Custom Equation
z = f(x , y)
= 1 a0 + a1*cos(w*x) + b1*sin(w*x) + a2*c

```
General model:
  f(x,y) = a0 + a1*cos(w*x) + b1*sin(w*x)
           + c1*y^2 + c2*y
Coefficients (with 95% confidence bounds):
    a0 =    91.99  (89.52, 94.46)
    a1 =   -11.71  (-12.54, -10.89)
    a2 =   0.4614  (-0.2845, 1.207)
    b1 =   -8.937  (-9.69, -8.184)
    b2 =   -5.566  (-6.095, -5.038)
    c1 = 0.006221  (0.005093, 0.007349)
    c2 =  -0.7534  (-0.8679, -0.6389)
    w =    0.2665  (0.262, 0.271)

Goodness of fit:
  SSE: 3.336e+04
  R-square: 0.8063
  Adjusted R-square: 0.8049
  RMSE: 5.708
```

```
library(ggplot2)

View(diamonds)
summary(diamonds)

summary(diamonds$price)
avgSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

q <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price", main="Diamond Pricing")
```

```
3rd Qu.:1.0400   Ideal    :21551   H: 8304   VVS2   : 5066   3rd Qu.:62.50
Max.   :5.0100                     I: 5422   VVS1   : 3655   Max.   :79.00
                                   J: 2808   (Other): 2531
    table           price             x               y               z
Min.   :43.00   Min.   :  326   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
Median :57.00   Median : 2401   Median : 5.700   Median : 5.710   Median : 3.530
Mean   :57.46   Mean   : 3933   Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
Max.   :95.00   Max.   :18823   Max.   :10.740   Max.   :58.900   Max.   :31.800

> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     950    2401    3933    5324   18820
> avgSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> q <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price", main="Diamond Pricing")
> q
> 
```

```
Data
diamonds     53940 obs. of 10 variables
Values
  avgSize    0.7979
  clarity    chr [1:8] "I1" "SI2" "SI1" "VS2" "VS1" "VVS2" "VVS1" "...
q            Large gg (9 elements, 3.3 Mb)
```

Figure 1: Matlab visualization (left) [54] and creation of a plot in R (right) [13]

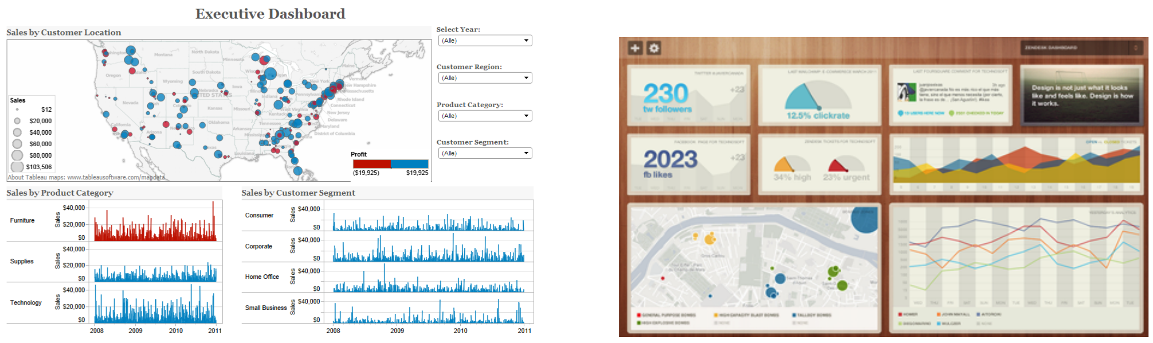


Figure 2: Tableau dashboard (left) [14] and Ducksboard dashboard (right) [11]

visualization. RStudio[2] is a frontend for the statistical language R[3]. R grew out of the discipline of statistics. It also has highly efficient data types, but is better suited to statistical modeling, and has better visualization tools. Both however have two major problems:

- They suffer from a very steep learning curve
- Both are not built around interactive or "live" data exploration.

Both languages and environments force a complex "Domain Specific Language" (DSL) which are hard to use without a background in computer science. Their DSL is highly specialized for the software's intended research field. Both languages can be used in an iterative environment and allow reevaluation of program blocks, but they are poorly suited for generating explorable or interactive visualizations (or more generically "encodings") of data.

In addition, data can not be linked on-the-fly and must be anticipated prior to rendering the visualization, which requires the user to understand the nature of their data and algorithms. This adds complexity to the task of visualization, but it of course allows the user to execute complex tasks in those languages.

The second figure (2) shows two tools that allow the user to interactively view (and in a limited way, even explore) data. "Tableau" (left) allows the user to build interactive "dashboards" in which data can be compared. However, this relies on a specific format of data. Users do not have the ability to actively change the behavior of a visualization or connect data freely without pre-processing. The right image shows "Ducksboard", a system that also allows interactive visualization and dashboard construction. Ducksboard allows for the aggregation of multiple streams of data (such as Facebook and Twitter), and allows visual properties such as color and shape to modified. However, as with Tableau, data are not linked and the user's ability to change the behavior of visualizations is limited.

None of these tools combines the aspects of interactivity, linking data together, options to explore and script the behavior of data visualizations (and thus build complex behavior), and working with heterogeneous data from different sources. The system that I propose in this work incorporates all these aspects.

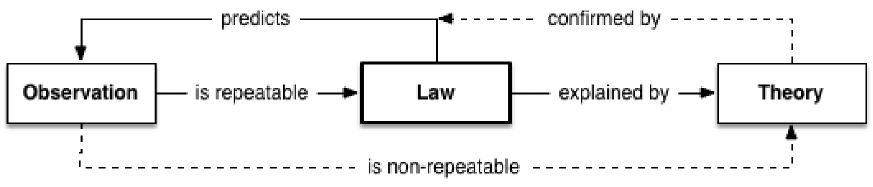


Figure 3: Scientific process to derive laws [55]

2.1 THE MIT AGELAB

The MIT AgeLab, "has assembled a multi-disciplinary team of researchers, business partners, universities, and the aging community to design, develop and deploy innovations that touch nearly all aspects of how we will live, work and play tomorrow"[1]. In this research lab, scientists from different fields perform experiments to investigate a wide variety of topics.

Typically, a data-analyst follows a classical investigatory approach to develop a theory or to find a law hidden in the data. Figure 3 depicts this approach. A scientist makes an observation during an experiment. Some of those observations might be linked by a law that predicts these observations. Those laws are then explained by a theory. Ultimately, scientists try to find those theories and their corresponding laws, which in turn rely on the possibility of repeated observations of patterns. This is where recorded data from experiments comes into play, which can be used to either confirm or falsify a theory. However, if this data becomes too large to analyze, the information (and consequently the knowledge) in the data may be lost.

The experiments performed at the AgeLab often produce enormous amounts of data. Typical experiments include subjects driving in a car for long distances over multiple hours while their physical fitness is monitored. The data generated includes physiological measures of heart rate and skin conductance, as well as metrics of driving performance such as lane position and GPS coordinates. Figure 6 illustrates the data-flow from an experiment into the visualization system. The sample rate of this data may be as high as 250hZ. Early workflows included Excel sheets and manual data inspection, but the sheer volume of data quickly overwhelmed these rudimentary methods.

Data visualization is one method of information extraction. A typical workflow is to integrate the raw data into a database management system (DBMS) and fetch data of interest as needed. The computer science experts at the AgeLab have created numerous scripts, written in the "Structured Query Language" (SQL) over the last years to fetch data. The data is then analyzed and visualized in different ways. Typically, data are visualized in R using the GGPlot[4] package, which is capable of drawing static images based on the underlying data. Other ways of visualizing the data include custom-made applications that visualize the heart rate of subjects or play the videos that were recorded during an experiment. Unfortunately some datasets are so huge that certain algorithms are needed which reduce the size of a dataset. Figure 4 illustrates a tool used for visualizing and manipulating the heat rate data of a subject. It features algorithms to extract so

2 Project website: https://www.rstudio.com/
3 Project website: http://www.r-project.org/
4 Project website: http://ggplot2.org/

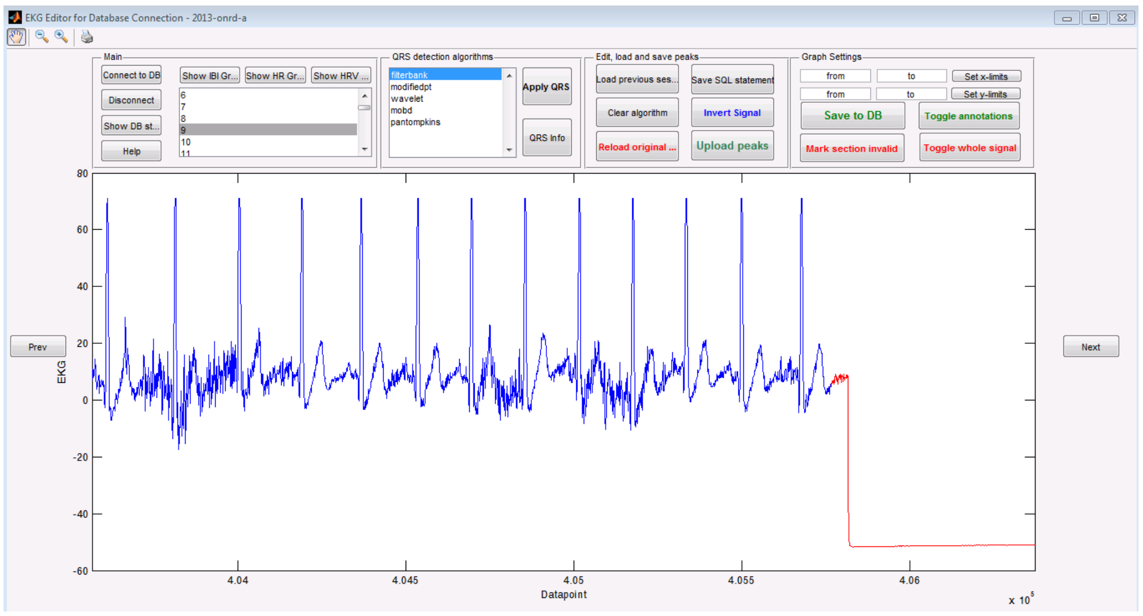


Figure 4: The heart-rate visualizer

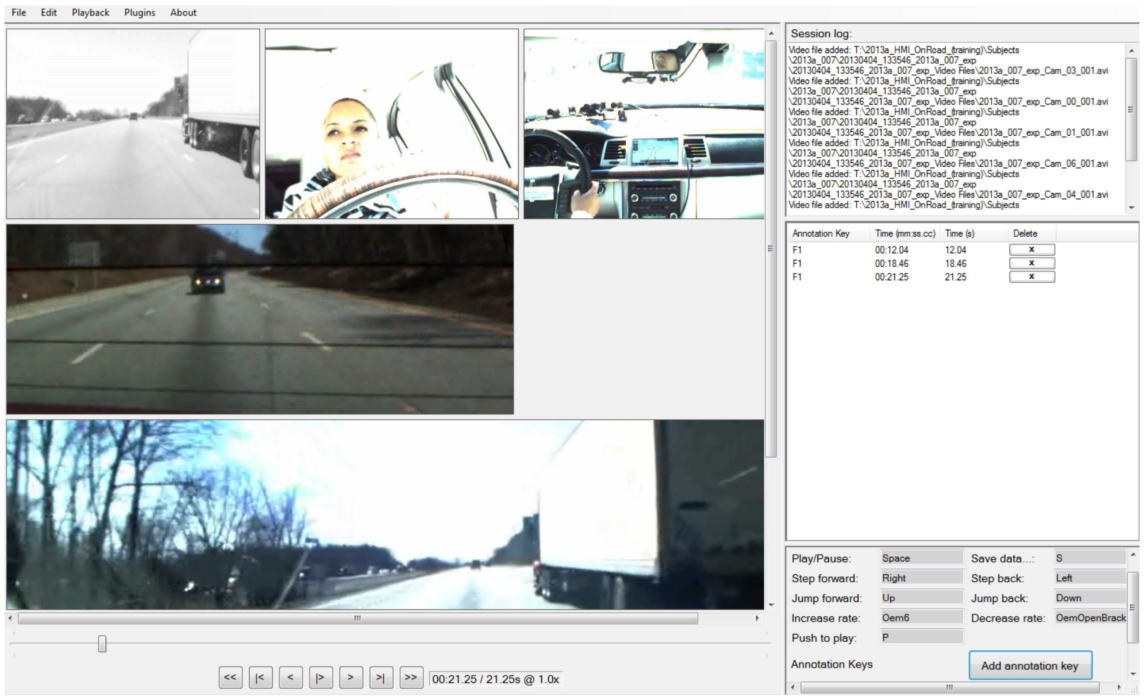
called feature points, data-points that summarize a small dataset, thus reducing the size of the data and easing the analysis. Figure 5 shows a system, again custom-made, frequently used to extract information from the data gathered during experiments. Videos are shown that were recorded during the experiment. The user is able to add annotations, such as "looked at mirror" to the video and save this information into the database. As shown, each of those applications focus on a specific task (images, video analysis etc.).

2.2 THE WORKFLOW

At the AgeLab, data analysts rely on data to either confirm an existing hypothesis or develop novel ones while looking at patterns in the data. Figure 6 illustrates how data flows. Starting at the subject, data is sensed and written into an on-board data-acquisition system. This data is then Pre-Processed and stored in a DBMS. Data queries are then written in complex SQL syntax and are imported into the tools and environments already discussed (denoted as "Pre-Processing" and "Analysis / Visualization"). Problematically, the complexity of the queries is often far beyond what the data analysts can work with without expert knowledge in that particular language. The proposed system tackles this with a repository of predesigned but configurable queries. Still, bringing datasets into new context is difficult, especially for users that do not have a strong background in computer science.

Many of the queries, aggregations and computations performed on the data exist solely to prepare one dataset for the context of another. The system described in this work offers a way to link data together easily, thus bridging the domains of "programming" and "analysis" of data.

Even if the data analyst creates a visualization that shows data in the context of other datasets, he or she is often still limited by the static nature of the visualization. As already stated, interactivity is a feature most publicly available visualization systems do not



Session log:

```
Video file added: T:\2013a_HMI_OnRoad_(training)\Subjects
\2013a_007\20130404_133546_2013a_007_exp
\20130404_133546_2013a_007_exp_Video Files\2013a_007_exp_Cam_03_001.avi
Video file added: T:\2013a_HMI_OnRoad_(training)\Subjects
\2013a_007\20130404_133546_2013a_007_exp
\20130404_133546_2013a_007_exp_Video Files\2013a_007_exp_Cam_00_001.avi
Video file added: T:\2013a_HMI_OnRoad_(training)\Subjects
\2013a_007\20130404_133546_2013a_007_exp
\20130404_133546_2013a_007_exp_Video Files\2013a_007_exp_Cam_01_001.avi
Video file added: T:\2013a_HMI_OnRoad_(training)\Subjects
\2013a_007\20130404_133546_2013a_007_exp
\20130404_133546_2013a_007_exp_Video Files\2013a_007_exp_Cam_06_001.avi
Video file added: T:\2013a_HMI_OnRoad_(training)\Subjects
\2013a_007\20130404_133546_2013a_007_exp
\20130404_133546_2013a_007_exp_Video Files\2013a_007_exp_Cam_04_001.avi
Video file added: T:\2013a_HMI_OnRoad_(training)\Subjects
```

Annotation Key	Time (mm:ss.cc)	Time (s)	Delete
F1	00:12.04	12.04	x
F1	00:18.46	18.46	x
F1	00:21.25	21.25	x

Play/Pause:	Space	Save data...:	S
Step forward:	Right	Step back:	Left
Jump forward:	Up	Jump back:	Down
Increase rate:	Oem6	Decrease rate:	OemOpenBrack
Push to play:	P		

Annotation Keys

Figure 5: The annotator

address. Therefore, the proposed system offers a way to not only interactively explore the data, but also to "play" data once it is visualized. For example, with such a system an analyst may virtually "re-create" a subject's driving session.

2.3 THE CONFLICTING FIELDS

As already stated in 2, the area of data visualization has no clear boundaries. This also implies that other fields reach into this domain, and that working on data visualization also means taking these fields into account. Problematically, some of those fields force constraints on the tools one can provide for the generic task of data visualization. At the heart of this dilemma are the classical non-functional requirements (NFR) shown in table 2.

Extensibility is crucial to tools that support a creative process. A data analyst needs to have the freedom to extend the system in the ways he or she envisions. This is supported in different ways in typical data visualization software. For example, R provides a library and package system that allows the user to load functionality on demand. The majority of R's packages are developed and maintained by its user community, thus extending R's analytical capabilities in ways that could not have been foreseen by its early developers. Similarly, the system proposed in this work makes very heavy use of extensions defined by the data analyst.

Portability is important in the special context of the workflows and environment of the AgeLab (see 2.2). The setups and operating systems of the lab's workstations are heterogeneous. Therefore, to reach a high adoption rate, portability is important.

When visualizing large datasets, performance is a key consideration. The single dataset used in this work (chosen from among hundreds available at the AgeLab) contains ap-

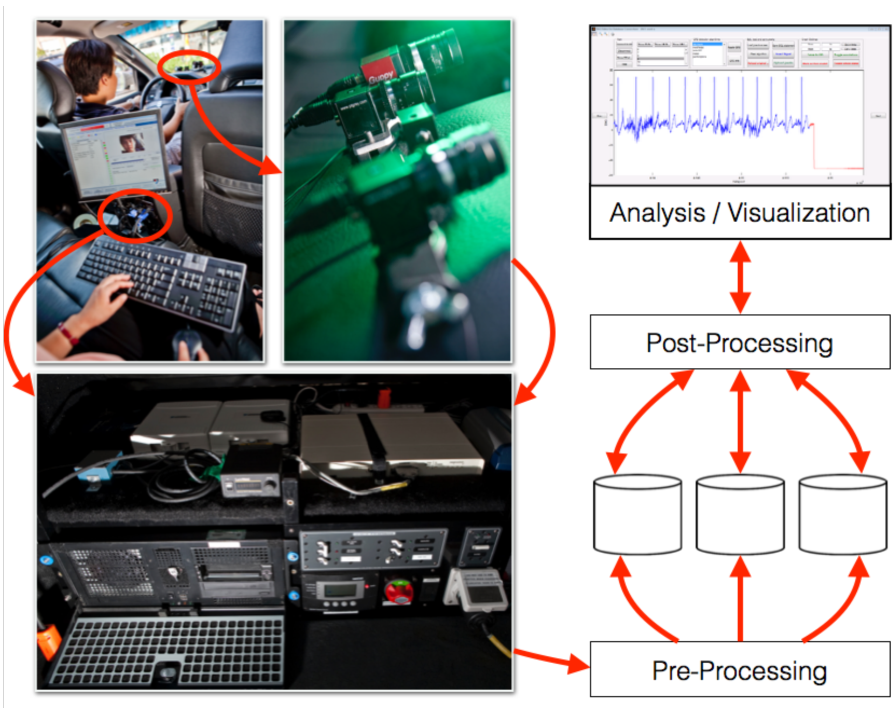


Figure 6: Data flow at the AgeLab

NFR	Description
Extensibility	A system design principle where the implementation takes future growth into consideration[73]
Portability	The usability of the same software in different environments[86]
Performance	The amount of useful work accomplished by a computer system or computer network compared to the time and resources used. [85]
Scalability	The ability of a system, network, or process to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth.[19]

Table 2: Table of conflicting NFRs

proximately 200.000.000 data points encapsulating measures along 20 dimensions and multiple terabytes of video- and audio-data. The algorithms and mechanisms used in the proposed system therefore have to take performance issues into account.

Scalability goes hand-in-hand with performance. Since the underlying hardware of a machine is the ultimate constraint of the performance of a system, the application needs to scale with the number of machines used. Scaling horizontally, or "scaling-out"[59], is quite popular since it is far more cost-effective and requires much less knowledge of the underlying system. Conversely, scaling upwards often requires specific operating systems and application architectures.

These NFRs need to be balanced. Extensibility might conflict with portability and performance if a developer wants to use a platform-specific extension or a low performance algorithm. Performance, on the other hand, can have a negative impact on portability since performance improvements might rely on platform-specific optimizations. This work is strongly based on cutting edge web technologies and attempts to balance the 4 NFRs.

DATA VISUALIZATION AND COMPUTATIONAL ANALYSIS As noted in 2, the question of "visualization algorithms and techniques" is naturally part of data visualization. This field focuses on the algorithms behind a visualization and the programs that define how data is translated into visual elements. This area focuses on analysis and computation based on the values of the data. The NFRs of this field (such as performance and fault tolerance) sometimes conflict with the targets of other fields. Making a complex visualization interactive is problematic if the visualization itself requires computation time that interrupts the users interaction with the visualization. This area also conflicts with the aspects of "big data", since a specific algorithm often requires certain data structures.

INTERACTIVITY AND WORKING WITH BIG DATA Even though the term "Big data" can fill multiple thesis on its own, it is of concern to the field of data visualization. This term refers to "a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications"[75]. The areas of user interface and interactive exploration are most affected here, since massive amounts of data need to be accessible via a suitable, responsive interface. Making such an interface interactive so a user can explore the information hidden in the data is particularly complex.

2.4 REQUIREMENTS

Paying close attention to the current workflow and the daily tasks data analysts have allow us to formulate a set of requirements. Many parts of the system were inherently undefined in the beginning. Together with the stakeholders at the AgeLab, a list of requirements was identified. Theese requirements are listed below. Even though this list seems small, a lot of work had to be put into envisioning how to realize those features in a user-friendly way. It is important to note that some of these requirements are fulfilled by some visualization system. However, they do not exist in one singular system.

010 Separation of concerns while creating a visualization

Separation of concerns is a method often used in computer science. It describes that "separating a computer program into distinct sections, such that each section addresses a separate concern."[89] In data visualization this approach is generally not used since a visualization often forms one single unit. With complex visualizations however, this becomes a valuable principle

020 Animations when switching between different datasets.

Animations are highly important to ease the perception of a complex visualization. It allows an analyst to perceive correlations easier and track constant parts in a visualization [animations]. This requirement is discussed in detail in chapter 3.

030 Highlighting parts of a visualization

To explore a dataset the designer needs to be able to focus on certain areas of a visualization. Highlighting refers to this ability. This requirement is discussed in detail in chapter 3.

040 Focusing and Linking

Focusing and linking allows a user to request details on demand. Focusing a certain part of a visualization leads to detailed information in other visualizations. This requirement is the reason for the requirement "Linking components". This requirement is discussed in detail in chapter 3.

050 Linking components and data

The linking of a component referrers to the ability to connect single visualizations with each other and enforce a logical link between them. This allows for example to sync to visualizations on the current time they show or show the same GPS coordinate.

060 Overlaying meta-information

This concept stems from [47]. Overlays can be put on top of a visualization to add more information or to make the perception of a visualization easier. Overlays include reference structures and summaries. This requirement is discussed in detail in chapter 3.

070 Annotating visualized data

Annotating data is crucial to enable analyst to be able to add meta-information which can not automatically created by the system. Adding meta-information to an existing dataset of a visualization is therefore a requirement.

080 Freedom of choice for libraries used to draw

Designers and analysts find themselves quite often in a "vendor lock-in" situation as most libraries for visualization are not compatible. Therefore, a system should allow the user to choose his or her preferred library for whatever task he wants to solve.

090 Creative freedom of how to draw

Additionally a analyst needs to have full control of the visualization itself. If he or she wants to change a single line in a visualization this possibility has to exist. The only way to allow this is to give the analyst the power over the programmatic structure of a visualization.

100 Sharing of workspaces

When creating complex visualizations reusability is an important issue. Other analysts have to be able to reuse visualizations that already exist.

110 Saving of results

When data is added during an annotation session, this data has to be saved int a SQL database.

120 Video recording of the workspace

Presenting the result of an analysis is highly important. Therefore, a modern visualization toolkit needs to be able to record a video from the interaction and the animated visualizations.

130 User-friendly access to the data queries

Querying data is often done in languages such as SQL. In many cases this language is only known by experts or computer scientists. A visualization system should give a novice user an easy entry into the system. The first barrier is to query for data. To not use languages such as SQL a library of predefined but configurable queries has to be created.

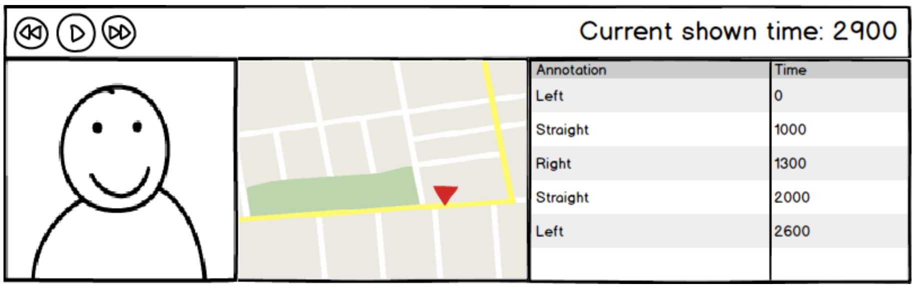
2.5 SCENARIOS

To showcase a few of these requirements a set of scenarios was developed. The stakeholders themselves were part of the creation of these visionary scenarios:

- Name: Annotate

  Summary: See a set of data dimensions and annotate meta-information.

  Description: Peter wants to annotate meta-information about the road to an experiment. Peter opens a workspace that shows the video recording, a map with the position of the car, and a table with the already added annotations. Peter starts to play the video. While the video plays, the position of the car on the map updates in real-time. Every time Peter presses a pre-defined button a new annotation is added to the table. This annotation has the same timestamp as the video-recording.



Annotation	Time
Left	0
Straight	1000
Right	1300
Straight	2000
Left	2600

Figure 7: Annotation session mock-up

Figure 7 shows a mock-up of how the user interface (UI) might look in such a scenario.

- Name: Patterns

  Summary: Compare two experiments and find unusual patterns.

  Description: Megan looks at the plot of the heart rate recording of a subject. She notices a peek when the subject was near an exit on the highway. Megan wants to understand if the exit was the cause of the peek. She opens a workspace showing the heart rate and a map with the current position of the car for two subjects. One of them is the subject that had a peek near the exit. Megan "plays" the heart rate. The heart rate and the corresponding position of the car on the maps update in real-time. Megan clicks on the heart-rate peek she saw before. The maps of both subjects jump to this position. The heart rate of the other subject is also updated and shows the corresponding heart rate. Megan sees a peek for this subject as well.

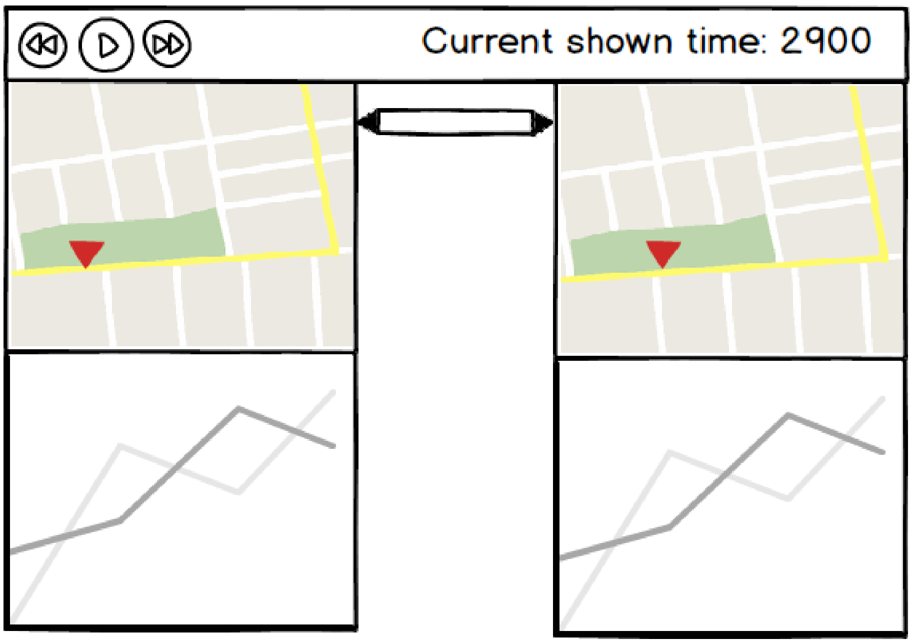


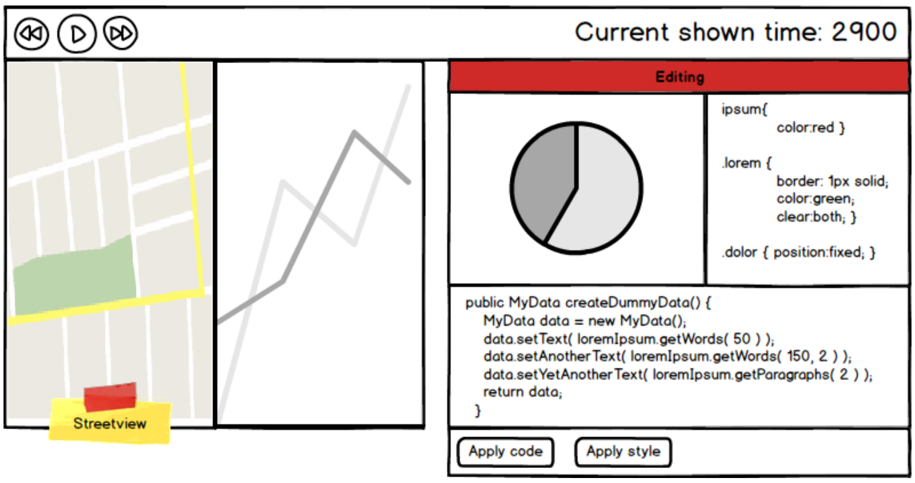
Figure 8: Pattern searching mock-up

In figure 8 an example is given for the UI during such a search for patterns.

- Name: Customization

  Summary: Create a customized graph for an annotation session.

  Description: Jon is trying to understand how fast the steering wheel is moved in certain situations. He opens a workspace that shows a map, the heart rate and the steering wheel position. Since the steering wheel only changes very little on a highway, Jon wants to modify how a steering wheel is shown. He opens a programming windows and a styling windows for the steering wheel and alters the visualization of the steering wheel to his needs. Jon also wants to see the surroundings of the car when the subject turned the steering wheel. Her opens the street-view in the map visualization to get a better understanding of the context.



```
ipsum{
        color:red }

.lorem {
        border: 1px solid;
        color:green;
        clear:both; }

.dolor { position:fixed; }
```

```
public MyData createDummyData() {
   MyData data = new MyData();
   data.setText( loremIpsum.getWords( 50 ) );
   data.setAnotherText( loremIpsum.getWords( 150, 2 ) );
   data.setYetAnotherText( loremIpsum.getParagraphs( 2 ) );
   return data;
}
```

Figure 9: Customized components mock-up

Figure 9 is an example for how the UI might look while the user is working on a component.

These three scenarios show how the implementation of the given requirements can lead to a system that allows an analyst to explore the accessable data. They will be used to evaluate the prototype in chapter 7.

3 RELATED WORK

Mike Bostock, inventor of the famous visualization library "D3" states that information visualization is still a "young and evolving" field [68]. He even goes so far as to say that we are only "scratching the surface of its potential". Even though this statement was written in 2008, it is still true. The last few years have been marked by several notable accomplishments in data visualization, such as the development and maturation of D3 and GGPlot. Still, the huge scope of data visualization leads to a slowly developing and evolving picture of this complex field. In the context of this work, several papers that target specific aspects of data visualization have been analyzed. It is important to note that those papers (and all papers that are cited but are not mentioned here) have had a strong influence on this work. Only a few of them, however, describe a complete system for data visualization.

One paper describes a system similar to the one proposed here, at least in terms of interaction, known as "SnapTogether"[62]. The authors propose a system in which different visualizations can be coordinated based on their data. The system's primary mechanism, called "snapping", allows the user to connect two windows based on a selected dimension of the data (such as time, identification number, etc.). Critically, the authors realize that data analysts will most often be working with correlated data that should be meaningfully connected as early in the analytical process as possible. Problematically, they implement their valuable concept of snapping and querying at a level comparable to visual programming and query building. The complexity and domain knowledge they therefore require from their users is a weakness of this system.

Another system, "imMens", described in [51], focuses on computation intensive querying and visualization techniques. The authors do not focus on the visualization itself, but present strategies to reduce and compute directly on the data that is queried. Their approach complements the "SnapTogether" system, since it acknowledges that querying and computing big data are highly complex tasks that are best hidden from the user. Even though this work does not focus on data reduction, [51] and their reasoning is part of the rational of this work. Other than these two rather complex systems, most of the papers analyzed fall into one of four categories:

SYSTEMS Papers in this category propose a system, meaning a set of "interacting or interdependent components forming an integrated whole"[21]. The two already cited papers fall into this category.

CONCEPTS AND THEORIES Papers describing an abstract idea of how visualization works or should be implemented.

ALGORITHMS AND FRAMEWORKS This category includes papers describing concrete implementations and functions used in data visualization.

DEFINITIONS Since many of the words used in the visualization field are not uniformly defined, many papers describe definitions and conventions on the semantics of words, such as "information" and "data".

3.1 ANALYSIS OF THE LITERATURE

In this sub-chapter, I describe the the publications that have most strongly influenced this work, providing short summaries of their hypotheses and proposals.

CONCEPTS AND THEORIES The paper "Big data Storytelling through Interactive Maps" [52] describes the approach Google has taken to make data visualization accessible for non-technical users. Most importantly, the authors acknowledge the need to interact with the visualization system without knowing specific DSLs. They also acknowledge the fact that context-specific visualizations, such as Maps, are of high value. This paper has influenced the design of a map-based view in this work, as well as an effort to abstract away DSLs wherever possible.

In [24] the authors describe the principal of "Focusing and Linking", which is foundational to coordinated views. "Focusing" means the selection of a subset of data, and "linking" means selecting the corresponding data in another view. This principal is the main idea behind the "linked components" concept described in this work (see chapter 5.3.1). Even though graphical queries are not possible and "area focusing" is not implemented in the prototype, the architecture supports these ideas very well.

The idea of redundant encoding using graphical overlays is described in [47]. The authors argue that "extracting, comparing or aggregating numerical values" is a task that not all visualizations are equally equipped for. Graphical overlays are a way to solve this dilemma, as they allow users to extract information with higher accuracy. The idea of context-specific and dynamically definable overlays in this work (see 17) is based on this paper.

ALGORITHMS AND FRAMEWORKS Software engineers have produced a huge set of patterns over the years. Consequently, there have also been attempts to find and abstract recurring solutions in visualization software. The authors of [25] and [38] describe a set of design patterns that is valuable for the construction of visualization software. Fundamental elements of the architecture of the present work are based on some of these proposed patterns.

A very influential work is "Data driven documents"[20], often referred to as "D3". D3 provides a library for data visualization and manipulation tasks. It is very flexible and incorporates a set of powerful and distinct technologies. The technical approach of the present work is based on the same essential reasoning as D3. In addition to the technical similarities of D3, [20] argues very persuasively that using complex DSLs for data visualization creates an abstraction gap between designers and developers. It is for this reason that the system described in this thesis does not make use of such DSLs and proposes to use different technologies. Even though this work does use libraries for data visualization, I am aware of the rather new and very promising concept of language-free dynamic drawing. This concept is outlined and also implemented by the remarkable Bret Victor [1].

The libraries used in this work often use animations in a specific visualization context. In [42], the authors describe "design principles for creating effective transitions" and showcase them on "common statistical data graphics such as bar charts, pie charts, and

1 Project website: http://worrydream.com/DrawingDynamicVisualizationsTalkAddendum/

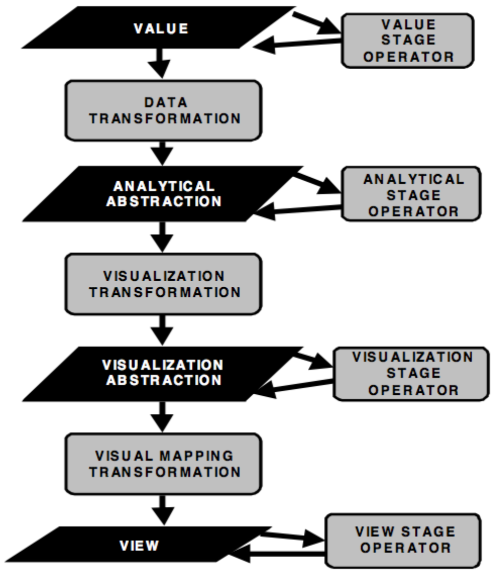

VALUE
VALUE STAGE OPERATOR
DATA TRANSFORMATION
ANALYTICAL ABSTRACTION
ANALYTICAL STAGE OPERATOR
VISUALIZATION TRANSFORMATION
VISUALIZATION ABSTRACTION
VISUALIZATION STAGE OPERATOR
VISUAL MAPPING TRANSFORMATION
VIEW
VIEW STAGE OPERATOR

Figure 10: Information Visualization Data State Reference Model[26]

scatter plots". Many of the libraries used in this thesis offer mechanisms described in that paper. This work, however, does not focus on these transitions per se, though the running prototype makes heavy use of them through those libraries.

DEFINITIONS In [26], the authors state that the terms "data, information and knowledge" are used "extensively, often in an interrelated context". They define those words clearly and propose to use this definition in data visualization in general. This document makes use of this quite helpful definition (see 1.3).

The "information visualization data state reference model" proposed in [27] has provided a guideline throughout the design process. The model describes the different states in which data can be visualized. Figure 10 depicts the model. The model is not directly used in the data-flow of the proposed system, though the model certainly influenced its design. The basic idea that data (here "value") moves from its original state to an "analytical abstraction" and a "visual abstraction" until it is finally shown in a view. In each of these states, operators may be applied. In the system I propose this concept is reused.

Part II

FOUNDATIONS AND DESIGN

4 DATA VISUALIZATION

As already noted, data visualization is a young field that has rapidly grown in importance over the last decade. Particularly in the last few years, this field has exploded in popularity due to open data initiatives and powerful tools that even non-technical users can easily use[61]. The exploding amount of available data, described in 1, is also one of the reasons for the popularity of the field. There are many explanations of why a visualization is more powerful than words, raw numbers, or tables. A very convincing one is given by Scott Murray in his book "Interactive Data Visualization for the Web":

"FORTUNATELY, WE HUMANS ARE INTENSELY VISUAL CREATURES. FEW OF US CAN DETECT PATTERNS AMONG ROWS OF NUMBERS, BUT EVEN YOUNG CHILDREN CAN INTERPRET BAR CHARTS, EXTRACTING MEANING FROM THOSE NUMBERS' VISUAL REPRESENTATIONS. FOR THAT REASON, DATA VISUALIZATION IS A POWERFUL EXERCISE. VISUALIZING DATA IS THE FASTEST WAY TO COMMUNICATE IT TO OTHERS." [61]

Murray describes the communication of information as the central purpose of data visualization and acknowledges the reduction of complexity as one of the designer's most important obligations. The author of [44] reduces this to the even simpler formula:

"THE KEY FUNCTION OF DATA VISUALIZATION IS TO MOVE INFORMATION FROM POINT A TO POINT B" [44]

Based on this description, one needs to define what "A", "B" and "information" are in this context. The word information is not easily defined as its nature is quite elusive but it most often refers to data that has been enriched with meaning [26] (as defined in 1.3). "A" and "B" refer to the roles of a human being in a communication of information. In [44] the authors describe two different kinds of data visualization based on the roles of points "A" and "B": **Exploratory** data visualization and **explanatory** data visualization. Exploratory data visualization refers to an instance of visualization in which data is moved from a dataset into the designer's own mind. The authors describe exploratory data visualization as "appropriate when you have a whole bunch of data and you're not sure what's in it. When you need to get a sense of what's inside your data set, translating it into a visual medium can help you quickly identify its features, including interesting curves, lines, trends, or anomalous outliers." In contrast, explanatory data visualization moves knowledge from the designer's mind to a third person, or as the authors put it: "explanatory data visualization is appropriate when you already know what the data has to say, and you are trying to tell that story to somebody else". Figure 11 describes "the nature of the visualization [based on] which relationship (between two of the three components) is dominant"[44]. All three roles need to be considered for a good data visualization. Choosing the dominant relationship defines what a data visualization communicates and how it can be used.

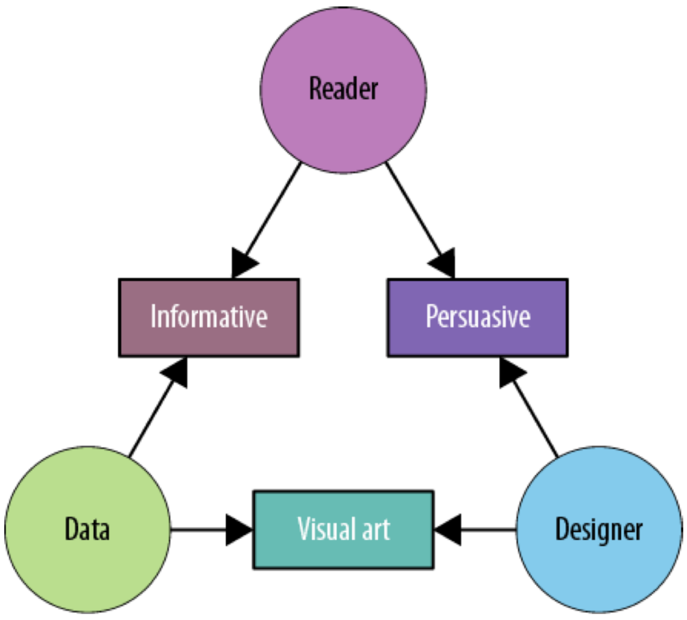


Figure 11: Relationships between the three roles[44]

A visualization is rarely purely "informative", "persuasive", or "visual art", but rather, is a mix of all these categories. Still, a designer should always ask him or herself which relationship is dominant in his/her specific use case. The system this work describes, outlines, and implements focuses on exploratory data visualization and is therefore focused on a strong relationship between the data and a reader.

4.1 THE VALUE OF INTERACTIVE DATA

Data visualizations have been static for ages. From primitive cave paintings through today's scatterplots and geographic maps, data is often visualized as a static image. Although certain interactive visualizations have been available for centuries, truly dynamic and widely available interactive data visualizations are the result of the software and hardware ecosystem of our modern world. The problem with static visualization is that "static visualizations can offer only precomposed views of data"[61], while interactive visualizations "empower people to explore the data for themselves"[61]. Interactivity is essential, especially the field of exploratory data visualization, which requires interactivity to identify the relationships between datasets. It allows an understanding of the data that is otherwise hard to achieve. As the author of [61] puts it: Users and analysts now can "overview first, zoom and filter, then [request] details-on- demand".

Many of the best practices in data visualization describe how data should be encoded. Chapter 4.2 describes this in more detail. Interactivity, however, is not an encoding. It is a way of looking and understanding the relationships in the data. There is not yet a conclusive approach to categorizing interactivity in data visualization. I therefore propose the following list of broad interactions:

HIGHLIGHTS The highlight pattern is quite popular in interactive visualization. It marks, annotates, or provides selected details for a specific subset of the visualization based on the user's behavior. Often, hovering with the mouse pointer over a certain part of a visualization triggers such a highlight.

ANIMATION Animations are moving parts of the visualization, and are used to establish spatial or temporal correlations or to ease the perception of the visualization itself. For example, a line chart may animate itself as it builds over time. Another popular example of animation is the transition between two kinds of charts. The authors of [42] even go so far to hypothesis that animations between certain charts give users a better understanding of both single and multiple datasets.

BRUSHING / FOCUSING AND LINKING The concept of brushing and linking, first described in [24], allows the analyst to first select a part of a visualization and propagate this selection to dependent, "linked" visualizations. This allows the analyst to view a dataset in the context of another. Linking mechanisms commonly operate on variables shared between datasets, such as time.

ZOOMING Zooming also refers to the selection of a subset of a visualization. The goal, however, is to request more details for this specific area. This mechanism is not only used to enable a better understanding of big datasets, but also as a performance tweak, since it avoids overdrawing a visualization that may have many data points. Interactive maps (such as those used for route planning) are an example of a visualization that is heavily dependent on the mechanism of zooming.

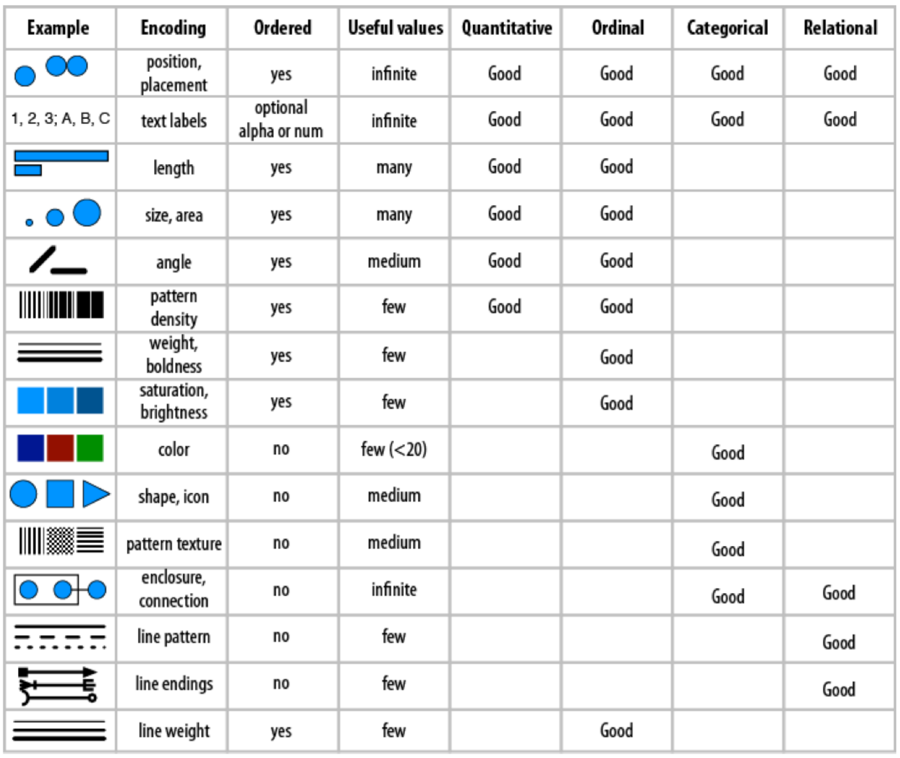
ANNOTATING The ability to annotate data is a rarely used, but sometimes a necessary pattern in interactive data visualizations. Users can add annotations to a visualization to store knowledge in a graph. There are no constraints on the appearance of annotations. They may appear as a pinned note, or a simple line of text describing points of interest. This pattern is distinct from the former ones, in the sense that it is the only one that enriches the visualization for subsequent use and is not strictly dependent on data in the source dataset.

4.2 ENCODING DATA

Information is always encoded in one way or another. In data visualization, data (and thus information) is encoded through a huge set of visual properties. Choosing the right one for a specific context is one of the tasks that makes data visualization complex. Different visual properties ("Encodings") can be used effectively to represent different forms of data, for instance Figure 12 illustrates dimensions often considered for different visual properties.

- Ordered: Ordering of an encoding refers to the ability of the human brain to create a natural ordering for said instances of that encoding. With ordered encodings, a human can decide if an instance of an encoding is smaller, bigger or equal to another.

- Useful values: The number of possible useful values for an encoding are determined by our ability to "perceive, differentiate, and possibly remember"[44] different values. This property is extremely important, as it allows us to extract the relationships between encoded data.



Example	Encoding	Ordered	Useful values	Quantitative	Ordinal	Categorical	Relational
● ●●	position, placement	yes	infinite	Good	Good	Good	Good
1, 2, 3; A, B, C	text labels	optional alpha or num	infinite	Good	Good	Good	Good
▬	length	yes	many	Good	Good		
• ● ⬤	size, area	yes	many	Good	Good		
∕ —	angle	yes	medium	Good	Good		
▥	pattern density	yes	few	Good	Good		
≡	weight, boldness	yes	few		Good		
🟦🟦🟦	saturation, brightness	yes	few		Good		
🟦🟥🟩	color	no	few (<20)			Good	
● ■ ▶	shape, icon	no	medium			Good	
▥ ▨ ≡	pattern texture	no	medium			Good	
● ●-●	enclosure, connection	no	infinite			Good	Good
┅	line pattern	no	few				Good
→	line endings	no	few				Good
≡	line weight	yes	few		Good		

Figure 12: Visual properties and their appropriate usage[44]

- Quantitative, Ordinal, Categorical, and Relational: These properties describe how well an encoding is suited to represent a specific type of data.

This list of encodings is in no way the only possible way characterizing encoded information, but it is one of the most well known. Visualization software typically allows the designer to either directly use such encodings by providing a language or drawing tool belt, or provide the user with a set of standard visualizations that can be backed by data (such as pie-charts, line-charts and heat maps). The system this work describes chooses to provide both to the user, as it is the only way to allow a novice user to visualize data and yet not trap an experienced user in a system that cannot accommodate complex visualizations.

4.3 CREATIVE FREEDOM

For some data, a simple representation might suit the purpose of transporting its information. A line-chart is probably sufficient to describe the revenue of a company (see Figure 13), as it is a simple, two dimensional dataset. However, one would be foolish to rely on standard visualizations to describe for example the complex set of intersecting lawsuits between companies. As shown in Figure 14, the visualization must adapt to the nature of the data. Even from these two relatively simple examples, it is easy to see that predefined visualizations cannot be used to describe every kind of data, especially when the data becomes more complex (for example, for the visualization of sequenced DNA). To be applicable to different data types, a predefined visualization needs to be

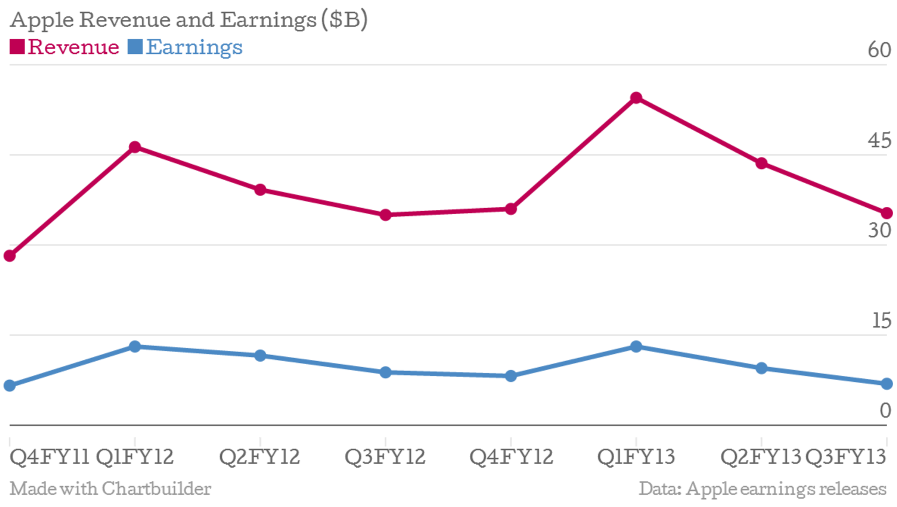


Figure 13: Line chart for product revenue[4]

very generic. This puts constraints on the information transported (such as information reduction if the number of dimensions is too small) and raises complexity not only for the data-analysts, but especially for the reader who cannot easily extract all the information he or she might need. Therefore, data analysts need to have the creative freedom to alter how data is encoded and visualized. This need for creative freedom is crucial to a powerful and useful data visualization system.

Many proposals have been put forth for how a designer can gain this power. A typical approach is to use DSLs that allow a user to describe a data visualization with a language that knows about concepts such as "lines" and other visual encodings. More promising is the recent development of libraries that do not utilize a special DSL for visualization, thus reducing the entry barrier for novice users. Such libraries do not rely on circles and pixels, but on a structural model that gets populated. Chapter 5 will describe in more detail how the system described in this work utilized this approach and achieves the goal of creative freedom.

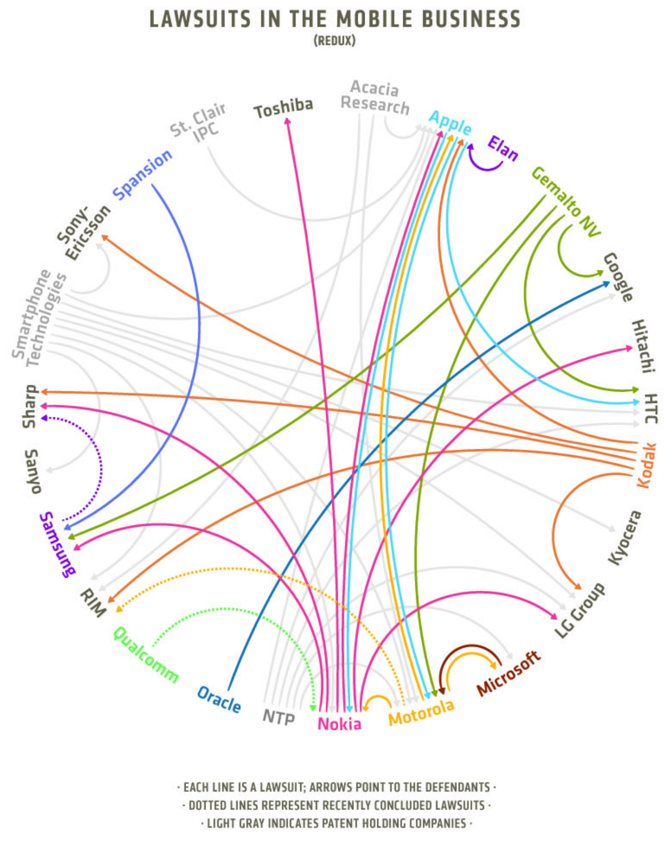


Figure 14: Complex visualization for lawsuits[7]

5 COMPONENT BASED VISUALIZATION

I propose to build a visualization system that uses "Components", not only for the underlying software structure, but also for of user-interaction. Those components can then host and control content, such as visualizations. How these components are built, how they interact with the user and each other, and how they use and process data is described in this chapter. The system I outline in this chapter is referred to as "DAVID", a system for "Dynamic Analysis and Visualization of Integrated Data".

Within the requirements that are strictly defined by the enclosing ecosystem of processes (such as limitations of the underlying operating system), component based data visualization leaves me with much room for creative approaches. An iterative development process is used to conquer this challenge, which allows me to examine different variations of the system. I build upon the idea of an evolutionary prototype [31]. This concept "acknowledges that we do not understand all the requirements and [we build] [...] only those that are well understood" [31]. Over the course of building the initial prototype, the different concepts outlined in chapter 3 have grown into a system that addresses many of the concerns listed in 1 and 2. However, the complexity of the system led to the decision to build a "horizontal" prototype. Horizontal prototypes are those that enable the developer to get a broad view of the complete system [87], without providing a complete subsystem implementation. Figure 15 depicts this idea. Horizontal prototypes provide an interface to all the system's intended features, even though a feature may not be fully implemented.

Though we provide a working system, it is built with the expectation that it will be extended later on. This is also a driver of the system's "library" based data fetching (see 5.4 for more details).

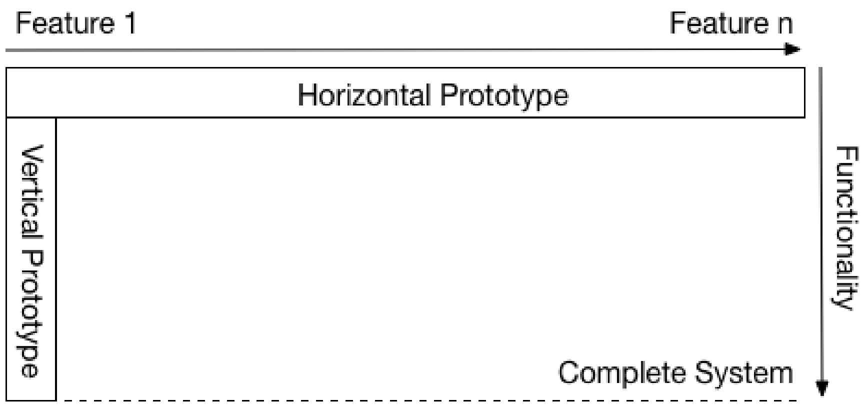


Figure 15: Horizontal prototyping

CRITERIA	RULES	PRINCIPLES
Decomposability	Direct Mapping	Linguistic Modular Units
Composability	Few Interfaces	Self-Documentation
Understandability	Small interfaces (weak coupling)	Uniform Access
Continuity	Explicit Interfaces	Open-Closed
Protection	Information Hiding	Single Choice

Table 3: Guidelines for building a module[57]

5.1 COMPONENTS AND MODULES

A component or module in software engineering is a single part of a system that may be loosely coupled with other components or modules [76]. The idea behind this concept is to separate program-elements from each other and therefore enable reuse. It also allows for more robust and flexible programs. Even though the terms "component" and "module" are often used synonymously, a module is designed based on 15 concepts, grouped into principles, rules, and criteria as defined in [57]. The authors of [57] argue that "a single definition of modularity would be insufficient; as with software quality, we must look at modularity from more than one viewpoint". The advantage of module-based design is to "separate the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality"[84]. The ultimate goal is to achieve "separation of concerns, and improve maintainability by enforcing logical boundaries between components. Modules are typically incorporated into the program through interfaces." [84]. The guidelines for building a module can be seen in table 3

Following these guidelines will ideally produce an independent module. Having an architecture in which modules comply with these guidelines also makes adding new modules easier.

This idea can be transferred to the domain of data visualization, in the sense that a visualization can be viewed as a single, self-contained module (since a data visualization is built by program code). Combined with a powerful architecture, such a construct allows us to conceive of visualizations as independent building blocks of a visualization dashboard for a given data set. This is the basic principle of what I call a "visualization component".

5.2 THE VISUALIZATION COMPONENT

The concept of self-contained modules adds structure to the otherwise loosely defined idea of data visualization. If a visualization is truly self contained, all its accompanying

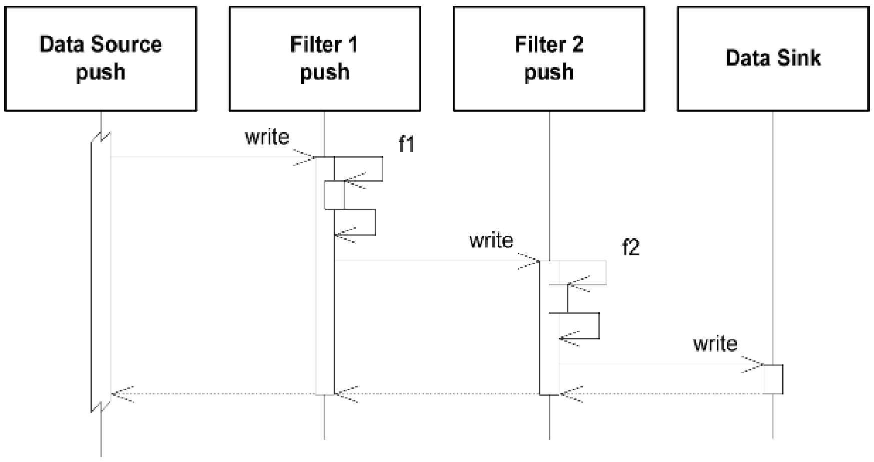


Figure 16: Pipes and filter[55]

parts are loosely coupled. This conception removes data fetching, data manipulation, and the user-interface from a visualization module.A visualization component is a specific "Viewpoint" of its backing data. Since the concept of a component is used as well, a data visualization component has a few, explicit interfaces that hide the information about the visualization algorithm and the internal data handling from the outside world. I define a data visualization component as

A SOFTWARE COMPONENT THAT OFFERS AN INTERFACE TO ALL NECESSARY METHODS FOR THE MANIPULATION, REDUCTION, PROPAGATION AND RENDERING OF THE VISUALIZATION'S UNDERLYING DATA.

This concept, however, requires the specification of a set of basic principles that define how such a component might work. First, how should the data be processed? Second, how should the interface to a visualization be designed or perceived, since, as stated in chapter 4 the freedom of creativity is crucial to good visualizations. That means that an interface must be visible to both programmers and end-users. This begs the question as to how a less technically minded end-user should interact with visualizations, and how the system can accommodate such interactions. Last but not least, component modules should follow consistent style conventions, without the overhead of defining every single property of an object for each part of the visualization. In the following sub-chapters we will examine possible answers to these questions.

5.2.1 *Pushing data*

In order to minimize relying on external data processing, a component requires that the outside system actively "push" data into the component. This concept is based on the "Pipe and filter" architectural pattern, which describes data as being passed from one filter to the next, each of which is connected by a pipe [55].

This architectural pattern also introduces the idea of an object primarily dedicated to the task of providing data. A component can thus be seen as a "Data Sink" that passively receives data from a “Data Source”, without knowing anything about this source or publishing any information (or requiring any specific interface or behavior) on the part of the “Data Source”. In this model, a visualization component behaves much as a reactive system does. In a reactive system, a program focuses on the data flow rather than related control structures.

The state of the data as it moves between the different parts of the system is based on the “Information Visualization Data State Reference Model”[27]. The raw data is altered and manipulated with certain goals, such as reduction or abstraction, before it is visualized. In chapter 6.2.2 I describe the technical aspects of the data-flow.

5.2.2 *Freedom through scripting*

A powerful visualization system must include standard visualization techniques that will work with diverse datasets “out of the box”. For example, a non-technical user may need to create a simple bar chart from an arbitrary table of data. While DAVID is able to provide this basic functionality, it also allows users to script the behavior of a visualization. The ability to dramatically alter the output of a visualization algorithm is a crucial requirement for a valuable visualization system. In chapter 2 I argued that languages typically used for data visualization (such as R or Matlab) give the user the ability to define custom behavior and manipulate the construction of visualizations. These languages achieve their flexibility via strong user communities that have created numerous reusable packages and modules that extend the base language’s functionality. Without this, a user is limited to the use cases envisioned in the language’s original construction. Such a lack of flexibility would represent an unacceptable constraint for the modern data-analyst, as the format and intended goals associated with any given dataset may remain unknown until analysis begins. A significant portion of data analysis is now spent on experimenting with and exploring the data prior to any finalized visualizations are constructed. This therefore necessitates an interactive interface that not only allows data exploration, but also direct manipulation of the behavior of the system itself.

One typical approach to these needs employs interpreted languages to re-evaluate parts of the analysis and/or visualization at system runtime. Most often, the interactive shells of these languages are the entry point for programmatic manipulation of the runtime behavior. The R statistical computing language, in concert with the well-known GGPlot package, is one of the most popular and powerful environments used for data visualization. This work is inspired by many of GGPlot’s foundational concepts and analytical strengths, but also attempts to address its shortcomings (as it generates static images). Rather than scripting only the visualization, another solution would allow the user to script the running system itself. I will use the term “Scripting Language” to describe

USER MANIPULATION OF THE BEHAVIOR OF A COMPONENT AT RUNTIME. Scripting languages are ideally suited to a component-based system. It is not unusual for scripting languages to provide an interaction layer between the language and its underlying system. This idea been used successfully in many fields (games, for example, often utilize the language "lua"[1] to achieve this goal[2]). However, this still requires us to define a suitable way of interacting with the inner parts of a component. Chapter 6 will focus on the details of this rather complex approach.

LIBRARY AGNOSTIC SCRIPTING Most languages ship with a default set of packages, and if those are not sufficient, many languages offer the ability to download additional packages. The implementation details of these packages systems vary, but nearly all languages allow it in one way or another. The component-based approach proposed in the present work allows us to choose a package based on the current needs of the data-analyst. This freedom of choice is crucial to a designer, as he or she must be able to use the right tool for the current problem. This is the reason for DAVID's "library agnostic scripting". DAVID comes with a set of powerful and widely used libraries. However, DAVID also allows the user to add whatever library he or she can find in its supported languages.

5.2.3 *Working with overlays*

An overlay is an addition to a dynamically defined visualization that adds enriching basic information . This allows a generic component to fulfill more specific needs, depending on its backing data. This implementation builds on the work of [47]. The authors of that paper argue that visualizations that are missing certain information can, through subsequent image analysis,be enriched with certain "overlaying" information. This concept is also applicable to standard data visualization. The authors of [47] argue that overlays fall into five categories. Figure 17 depicts these categories on a generic bar chart. Overlays fall into one of five categories:

REFERENCE STRUCTURES These provide guidance for the human eye and try to address the shortcomings of some visualizations (as discussed in 4). For example, as described in [29], size may be hard to estimate, depending on its context (for example, in pie charts). Reference structures help the human eye to compare values.

HIGHLIGHTS This overlay is used to focus the viewer's attention on a specific point. This is especially useful for interactive visualizations, since more than one aspect of the visualization may change simultaneously.

REDUNDANT ENCODINGS Similar to reference structures, redundant encodings illustrate values in alternative ways so they can be perceived easily (i.e., shape and color being used to differentiate identical categories).

SUMMARY STATISTICS Summary statistics provide the viewer with salient metadata regarding aggregate or group characteristics, and are often used to illustrate patterns in the underlying data.

ANNOTATION Annotations add information which is not present in the raw data to the dataset. In the context of interactive visualizations, this can mean displaying additional information and/or providing the capability to add such information to

1 Project website: http://www.lua.org/
2 World of Warcraft for example (http://www.wowwiki.com/Lua)

a visualization. As argued in 2.2, this is also a requirement for the system proposed in this work.

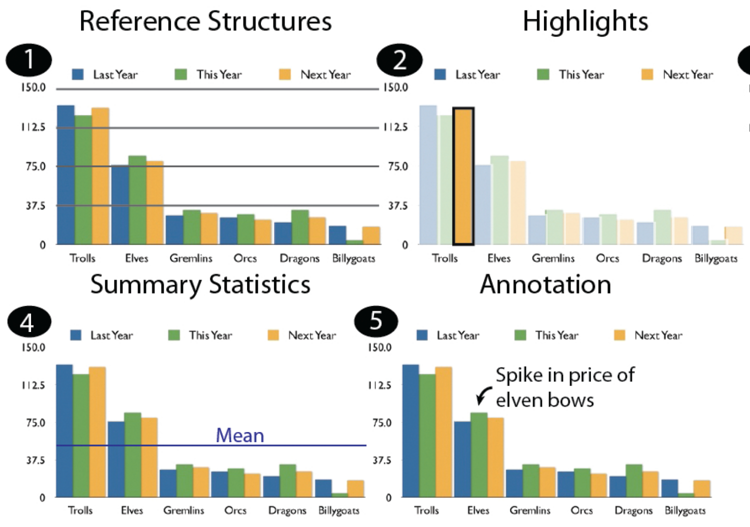


Figure 17: Overlay categories[47]

The reference implementation discussed in 7.1 shows examples of implemented overlays.

5.2.4 *Shadowing visual styles*

A basic principle in computer science is the "separation of concerns". The principle states that a program should be divided in such a way that each part addresses a different concern. When transferring this principle to the domain of data visualization, one can identify three separate concerns:

- Describing the structure of the visualization
- Describing the style / appearance of such a visualization
- Describing the logic and behavior of the visualization

This principle means that a visualization is not only a pixel-based image, but also has a model defining its underlying structure and behaviors. In a bar chart, that structure would be the bars itself, the labels of the bars etc. This model, however, is separated from the visual representation. Bars can appear in different colors, or with a variety of embellishments, such as borders and shadows. A system built on the principle of separation of concerns should provide the ability to alter the visual properties of the visualization without recreating or altering the model or its behavior. The goal is to dynamically change the representation during runtime, thus giving a user the ability to work with an iterate the aesthetics of a visualization. Such changes should only apply to a specific visualization, and not the visualization's general class. However, to allow the user to work in a consistent way with all components, the interface with which a user defines these properties should always be the same. I propose transferring the concept of

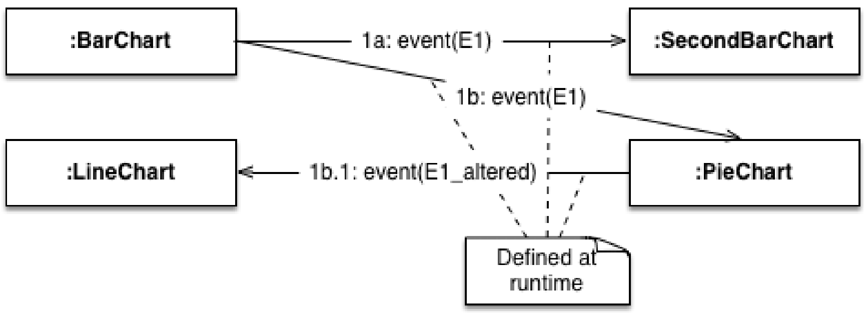


Figure 18: Flow of events between components

Cascading Style Sheets[3] to the domain of data visualization (as showcased in the popular d3.js library[4]). This allows the user to work with a consistent and proven backing model (the Document Object Model[5]), but at the same time allows the user to build upon the flexibility that the separation of model and visual properties provides. Learning about a domain-specific model is thus not required. The barrier to entry for using this styling system is therefore quite low.

5.3 TALKING COMPONENTS

As stated in 3, one criterion of a component is its composability. In software engineering terms, composability is defined as the ability to create "software elements which may then be freely combined with each other to produce new systems"[57]. Transferring this concept to the domain of data visualization means that components should be composable on the code level and at the level of the user interacting with the components. To achieve this I propose the idea of "talking components".

Components should be able to exchange state information and notify each other based on their internal behavior. I suggest combining the concepts of event-driven systems, which are characterized by the fact that "execution is in response to events happening while the program is executing"[49] and flow-based programming, in which components "exchange data across predefined connections by message passing, where the connections are specified externally to the processes" [80]. This means that components would not pass data between each other, but instead primarily send events between each other based on the externally defined connections. Figure 18 illustrates this idea. The connection that describes how information flows (based on flow-based programming) is defined at runtime. Events are then passed over this connection while the visualization is active. This implementation differs from event-driven programing in that a component cannot register for arbitrary events and, even more importantly, can alter the event once it arrives at a component. This allows for a particularly dynamic composition of components and ways of "talking", as each component can add its own understanding of an event and how to pass it. This is particularly helpful when propagating changes that happen during interaction with a visualization. For example, if a scale changes on one visual-

3 Standard website: http://www.w3.org/Style/CSS/Overview.en.html
4 Project website: http://d3js.org/
5 Standard website: http://www.w3.org/DOM/

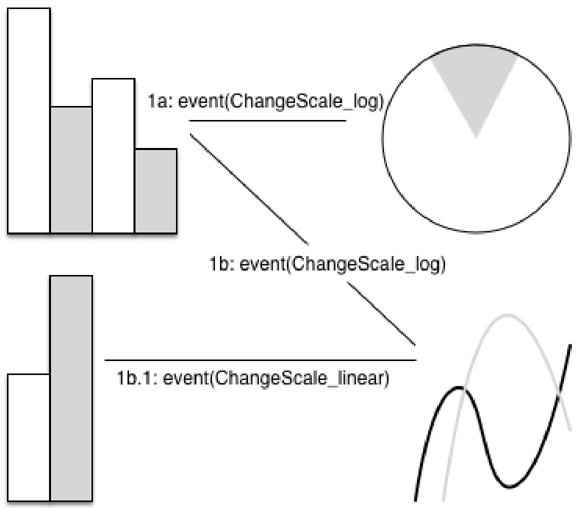

1a: event(ChangeScale_log)
1b: event(ChangeScale_log)
1b.1: event(ChangeScale_linear)

Figure 19: A set of components exchanges event information based on their defined flow

ization, this change can be propagated to a set of user-defined components. Figure 19 showcases this. Notice that this concept is near to “reactive programming”, though it is not the same, as the propagation of events and the external specification of connections is not considered a part of the reactive paradigm.

5.3.1 *Linking data*

As argued in 1, multiple data streams are now often linked together. However, the links between data sources are not always clear. Linkages may be communicated implicitly through the dimensions of the data and their scales, or they may even be based on secondary computations. Bearing these complexities in mind, data linkage makes component linkage more powerful. It allows us to view data in the context or perspective of a linked dataset (and its corresponding visualizations).

An interactive visualization allows the user to explore different attributes of the underlying data or the visualization by manipulating predefined interface elements, such as hovering over a data point to reveal its associated information or zooming into a map to reveal greater detail. The data the AgeLab uses is rather typicall for multi-sensor data in that all AgeLab datasets share the dimension of time (in other words, every data point in every dataset has a timestamp that allows it to be contextualized with other data points). One of the major problems with “timed” data is that it is hard to experience such data “live”. Seeing how values change over time is often more informative than looking at static visualizations of discrete time periods. Therefore, DAVID introduces the ability to “play” data. If a dataset has a time-based value, DAVID can use this value to visualize several dimensions over time. Changing values are animated, which allows the viewer to understand and perceive the differences, development, and overall patterns in the data [42]. For this reason, DAVID links all datasets on the time variable by default.

Figure 20 depicts a general abstraction of the types of linking, the GPS linking and the time linking. Both are a good example for how different kinds of data can be connected.

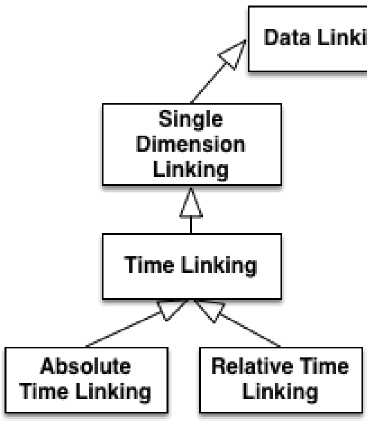


Figure 20: Hierarchy of linking types

Figure 21 shows how these different kinds of linking can be used to connect two components. Component l1 is linked to m1, showing the same time. m1 and m2 are linked using GPS coordinates. Therefore, they link on their coordinates. m2 and l2 are then again linked on time. This allows to see l1 and m1 in their own time context, which does not need to be the same as for m2 and l2. To link the l1 m1 timeframe to the l2 m2 timeframe m2 needs to be able to translate the GPS coordinates it receives from m1 to its own time-dimension.

Since time is known as a dimension, a user only needs to provide a function that is able to translate from time to dimension x, rather than to introduce a way to understand the dimension. The formula listed below deescribes how each component can use time as their basic linking dimension.

$$f(x_1)=y; x_1 \epsilon\ Time; \underbrace{y \epsilon M_{valueDomain}}_{\text{an intermediate representation}}; g(y)=x_2; x_2 \epsilon\ Time$$

A function $f(x) = y$ where x is the time and y the data one wants to link, translates between the data domain ($M_{valueDomain}$) back to the domain of time. As shown in figure 20, time itself is also part of the data domain and therefore time can be linked with time itself. In such a special case, this linkage is called “syncing”. Only these two functions are necessary to link two datasets that share at least one dimension. It is not even necessary for both datasets to contain a time dimension. The functions $f(x)$ and $g(y)$ can be used

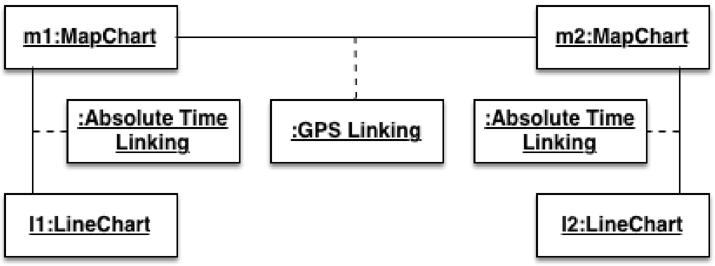


Figure 21: Using linking types to connect different kinds of data

for separate linking operations. But to ”play” live data both datasets must expose a time dimension. The only exceptions are datasets where the resulting values are unique. If that is the case, DAVID would not need the translation function $g(y)$, since it would know on which value to sync.

At least one of these linking functions is provided for each data call, as enforced by the system's underlying architecture. See chapter 6 for more details.

5.3.2 *Linking components*

Usability is one of DAVID's most important non-functional requirements. When it comes to working with data, the analyst, requires freedom of interaction with the data and an easy way to solve recurring problems. Data linkage is one such problem. As described previously, data can be linked via suitable functions. Such functionality should not be restricted solely to the programmatic level. Linking visualizations must be made easy, as it abstracts away rather complex data management operations, depending on the data. This is why DAVID uses visual linking of components to define how the backing data are linked. A user can simply draw a line from one component to another, thus initiating the linking of the data. Since the system provides at least one of the two functions ($f(x)$ and $g(y)$), such links can always be established. A user might later change the meaning of the drawn line or remove it entirely. To some extent, this allows the user to visually program parts of DAVID. The concept on which this idea builds is described in the beginning of this subchapter (see 5.3).

5.3.3 *Network-wide events*

Since relatively lightweight "events" are used to communicate between components, it is easy to extend this mechanism to more complex scenarios. For instance, events may be moved between machines in addition to between components, since the amount of data transmitted is small and the functions on which data are linked can be defined separately for each machine. Network-wide events only assume the existence of a component that is capable of using a socket connection, rather than a direct connection to another component. As shown in Figure 18, a component is by definition able to alter the events it receives. Components have a strong influence on how and which events (and data, if needed) will be passed to another component. A component that connects to another machine simply uses a different communication technique that is hidden from the user.

5.4 DATA DOCUMENTS

Pipes and Filters, as shown in figure 16, receive data from an object called a "data source". The visualization components also require such a source. This data source abstracts away the complexity of data fetching and pushes the received data into a component. Before we can define the system around those components, we must carefully consider how such a data source might be structured.

Multiple papers referenced in chapter 3 investigate how working with data sources might be implemented. A common approach is to transfer select aspects of the underlying database into the application. The authors of [62], for example, argue that a visual interface for creating common database operations such as joins and selects is a suitable and powerful method for this purpose. In [38], patterns are introduced that allow a mapping between relational database concepts and a program. While this may be a valid approach when the concrete database system or architecture is known, the concept does not generalize to all situations. In contrast to those concepts, I propose the idea of "data documents" that base their content on "library based queries" which separate the data fetching from its specific storage method.

Data documents are simple data wrappers which are backed by a powerful library of available queries. How the data source and the visualization might interact with such a library, and how a library can be adjusted to its context, is discussed in the following subchapter.

5.4.1 *Library based queries*

Reduction of complexity is a typical goal of data visualization. In the domain of data acquisition, this goal also applies to the retrieval of data. The importation of data into the visualization system is one of the most complex and potentially time consuming tasks faced by data analysis. Tools such as Matlab or R allow the user to connect to databases and fetch data, but there is no guarantee that the analysts will possess the skills necessary to write the queries necessary to load the desired data. Due to the sheer complexity involved in data fetching and the accompanying DSLs (such as SQL), I propose to shift away from a specific data fetching mechanism and instead advocate for "Library based" queries.

Library based queries are simply a collection of predefined data fetching methods, which is in turned made available to the user of the data visualization system. Collections of fetching methods must allow some flexibility, in the form of customizable query parameters, but nevertheless this reduces the complexity of a query by abstracting away the underlying DSL that is used to fulfill the request. Such queries must indicate how the resulting data is organized. Since the user is not supposed to interact with the fetching mechanisms directly, the library must define both how the data are obtained and preprocessed. Preprocessing in this context means organizing the data in a simple, useable, and generic format for subsequent operations. The table based "Comma Separated Values" (CSV) format is one such organizational schema.

5.4.2 *Promise based data*

Data documents are not aware of the specific query actions that produce their underlying data. This allows for a separation of visualization and data, but as a consequence, we have no knowledge about when data is returned or updated. A query might, for example, use a single web-request to produce data, but another query might return a stream to a file, constantly returning updated or additional values. To tackle this awareness problem, we may use "promises". The concept is proposed in [17] and describes "an object that acts as a proxy for a result that is initially unknown, usually because the computation of its value is yet incomplete"[81]. Following this idea, the queries a library offers can return the promise to deliver a value at some future point in time. Once this value arrives, regardless of how it was generated, the data document is able to store this value and push it to the component (see 5.2.1). The concrete architecture, especially how data is fetched and processed is described in the following chapter.

Part III

SOFTWARE DESIGN AND IMPLEMENTATION

6 SOFTWAREDESIGN

The concepts that lie at DAVID's heart raise the question of how such a system would work in reality. This chapter is the blueprint for DAVID's prototypical implementation, and function as a proof of concept and a way to evaluate the proposed solutions for the common problems faced by modern visualization software and languages. This chapter describes DAVID's different parts from an architectural viewpoint and highlights some specific implementation decisions that support some of DAVID's most crucial features (such as styling of visualizations, library based queries, and scripting).

6.1 ARCHITECTURE: THE BIGGER PICTURE

Figure 22 illustrates DAVID's essential elements. The diagram shows the most important components, each of which is organized under "interaction", "logic" or "data" packages. This distinction follows the well-known MVC ("Model, View, Controller") pattern. A data analyst may want to look at the same dataset in different ways. As already discussed in 5, data should be accessible from multiple viewpoints, meaning that the same dataset can be used in different contexts, even at the same time. The MVC architectural pattern allows us to address this need. MVC is "useful when the same model element has to be presented in different contexts simultaneously" [23]. MVC separates the system into 3 distinct groups of elements (taken from [23]):

MODEL OBJECTS represent the application's domain knowledge. All elements that are necessary for fetching data, selecting data, or preparing data are part of the model.

VIEW OBJECTS visually represent the model and provide an interface for interacting with it. For example, a view object might be a control panel, spreadsheet viewer, or a system or buttons and toggles. In DAVID, I also include interaction logic in this group, though it does not alter any data in the models.

CONTROLLER OBJECTS link model objects with their views. They provide the underlying logic, checks, and safeguards that allow the model to be manipulated or data to be edited via the views.

One unusual deviation from the MVC framework is the addition of scripts that run only within the front end and work with data representations. However, those script do not affect the the other packages. Any scripts that affect components outside of the interaction package would need to do so through controllers.

The logic package groups controllers and libraries that can be used to alter or enrich data, such as annotations and standard metrics which translate data into a predefined format. Finally, the data package is very similar to the classical idea of a model group. Sub-packages contain mechanisms to fetch and store data. These packages form the core of DAVID.

Figure 23 shows which elements of DAVID's architecture correspond to the MVC paradigm. DataDocuments are considered to be models, since they hold and manage

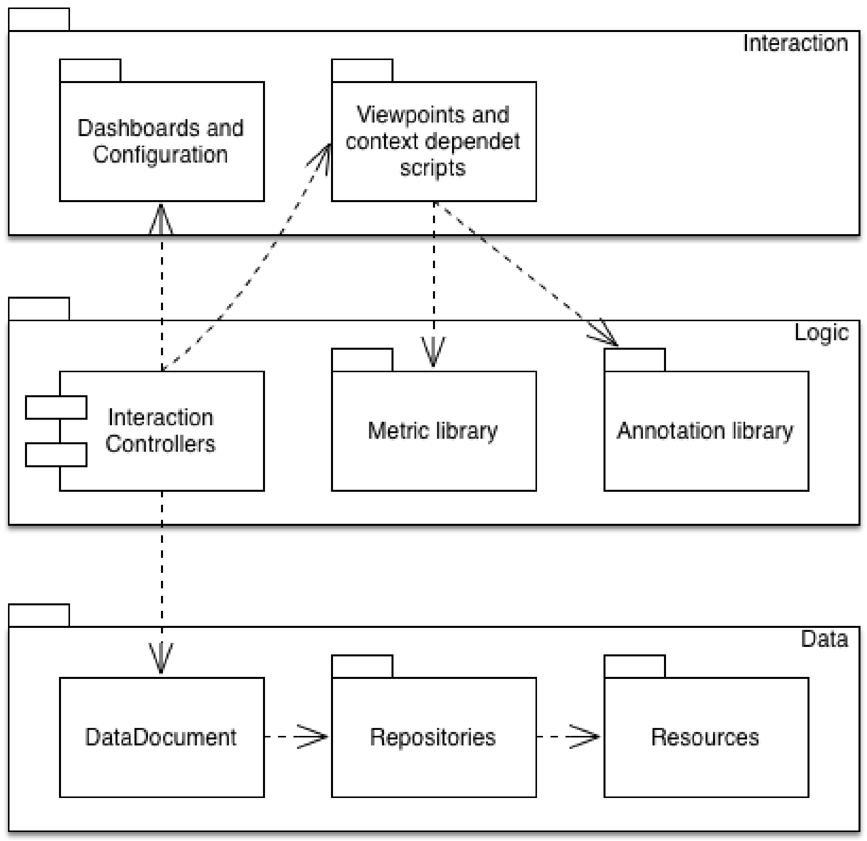


Figure 22: Main packages comprising DAVID. Dashed lines mean "usage"

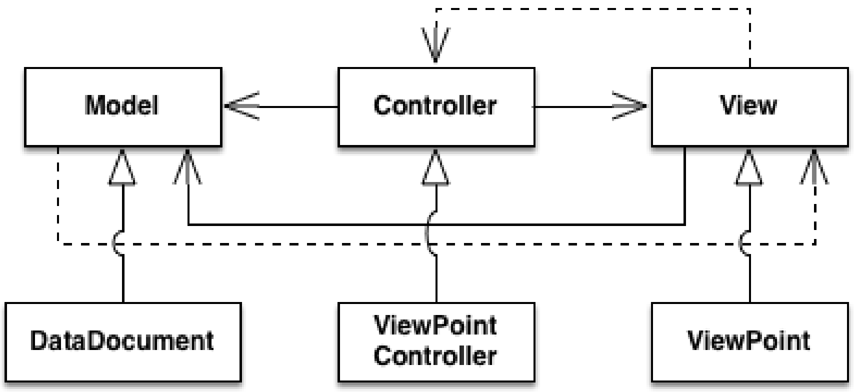


Figure 23: MVC analogies

all data that is altered, used, or shown in the system. ViewPoint Controllers are the controller elements, and only appear in conjunction with a View. Views are implemented through ViewPoints, which allow a specific view of the data. Figure 24 provides a more detailed description of the corresponding classes. Controllers are not implemented as a single generic class, but instead consist of a value object which is created on the fly. This decoupling of the controller and its parameters allows the user to dynamically build their own dashboards based on self-defined values. Controllers are also able to communicate with each other. Such a connection is necessary because controllers might appear on the same screen (depending on how the user chooses to build the dashboard) and they need to share data or states (as described in 5.3). The ViewPointValues follow the interface definitions of the "ViewPointObservable" interface, which allows classes to observe one another. All ViewpointValues manage a collection of connectors, which in turn defines which two Observables they connect.

Views consist of the ViewPoint, their ViewPointOverlays(which Views create and execute based on the backing model), and a ViewPointComponent, which manages the visual states of the ViewPoint. These states are the classic UI interactions, such as minimizing the ViewPoint, or moving it in space. The ViewPointComponent does not correspond to the specific data visualization. The visualization is handled by the ViewPoint and its logic.

As shown in figure 24, all ViewPoints rely on a DataDocument. DataDocuments have a supporting set of classes (see 6.3) that are responsible for providing interfaces for data-calls, fetching of data, and pushing the result to the ViewPoint. DataDocuments, in turn, rely heavily on Repositories. A Repository is a class that simply offers a collection of data-calls to the user. In chapter 6.3, I describe in more detail how these Repositories work.

DAVID uses the angular.dart framework[1] to implement MVC concepts. Matthes [55] defines a framework as "a set of classes that embodies an abstract design for solutions to a family of related problems, and supports reuses at a larger granularity than classes". Systems that are designed to be used at this larger granularity often introduce some kind of control flow. In this case, the "Hollywood principal" is used to define how com-

1 Project website: https://angulardart.org/

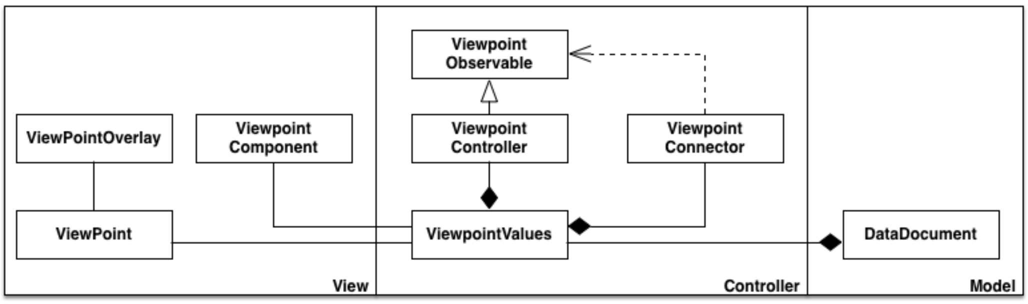


Figure 24: Model view controller view of the system

ponents are constructed and how the control flows. The Hollywood principal "invert[s] the control flow of the application" and brings the view into existence. Figure 25 depicts this principal. The framework (blue) monitors the control of the flow of execution, only sometimes calling certain components or emitting events.

In black box frameworks, the application only reacts when certain hook methods are called. In contrast, white box interfaces provide certain classes implementing a specific interface [55].

Angular falls into the category of white box frameworks, as it works primarily by inheritance. Angular auto-generates the code that glues together the controller and the view, as defined by the developer. Figure 26 depicts this complex but powerful structure. Controllers (blue) are typical classes that expose certain methods and variables. A view can then use those exposed elements through the automatically generated scope.

In DAVID, users are allowed to build their own dashboards consisting of many ViewPoints of data. Dashboards display multiple elements on a host screen and have, for example, been used by Apple to show independent, small widgets in OS X, and by Google to show selections of personal data on a website. The design of a dashboard and its components must be flexible, since the final appearance of the Dashboard is unknown at compile time . Once a user has defined the dataset he wants to look at and the ViewPoints he wants to visualize, a screen has to be built that is capable of displaying this selection. In DAVID, a dashboard is called DataView. A DataView collects sets of DataDocuments,

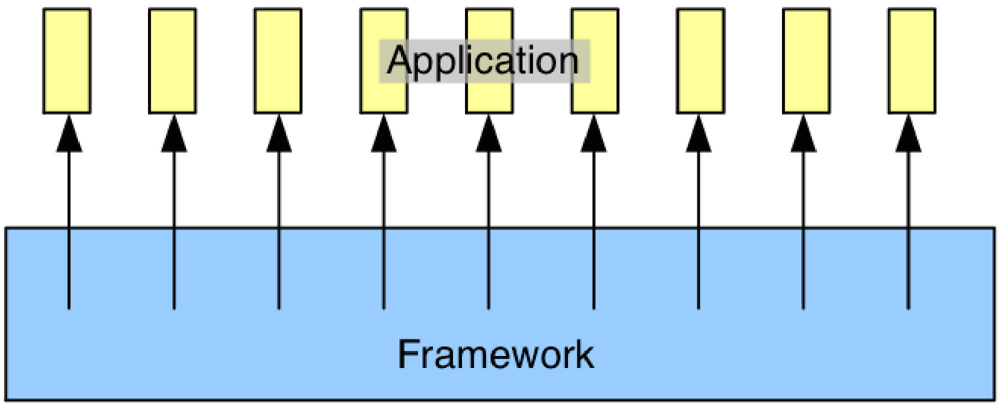
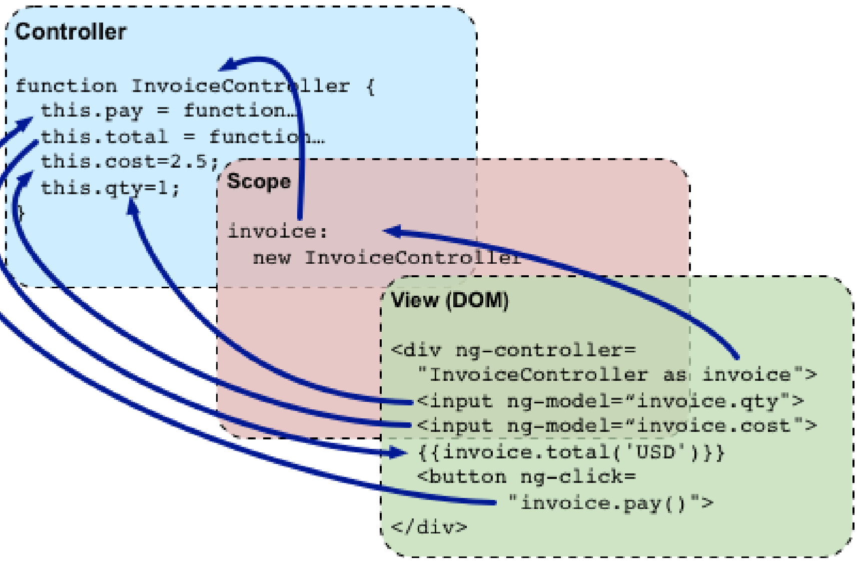


Figure 25: Hollywood principal (based on [55])



```
function InvoiceController {
  this.pay = function…
  this.total = function…
  this.cost=2.5;
  this.qty=1;
}
```

```
invoice:
  new InvoiceController
```

```
<div ng-controller=
  "InvoiceController as invoice">
  <input ng-model="invoice.qty">
  <input ng-model="invoice.cost">
  {{invoice.total('USD')}}
  <button ng-click=
      "invoice.pay()">
</div>
```

Figure 26: Structure view of MVC based scopes in angular[10]

which in turn know about their ViewPointValues (see figure 24). A DataView first loads all DataDocuments that are assigned to this view and then uses the ViewPointValues to create the corresponding ViewPointController. This creation is handled by the angular framework.

Deployment

DAVID can be deployed in a variety of ways. The system itself and its core components run inside the Dart virtual machine. As depicted in the deployment scenario in Figure 27, DAVID can be deployed independently of a database location. On the right side of Figure 27, DAVID is deployed on one machine. The "native shell" provides file-system access for DAVID, which is used by a specific mechanism in the "resources" package. This package holds all concrete implementations of data fetching mechanisms. The distinction between data and the mechanisms to fetch it makes it possible to deploy this part of DAVID on other machines. The native shell allows DAVID to use OS-specific mechanisms, such as a SQL driver. On the left side of Figure 27, DAVID is split across two machines, one running the system and one running only the native shell. This way, DAVID can access a central repository of files. The combination of this unusual deployment mechanism allows DAVID to choose how and from where data is fetched. In this way, the system achieves true location transparency.

It is important to note that this separation, and the possibility of two instances of DAVID communicating with each other, allow the system to scale horizontally. Two instances can communicate while visualizing. Theoretically this allows to visualize data in a way that would have exceeded the capabilities of an isolated machine.

Enforcing separation of concerns

Software architectures tend to rot over time[53]. When parts of a system change, especially if such changes occur often and rapidly, this can cause problems with further

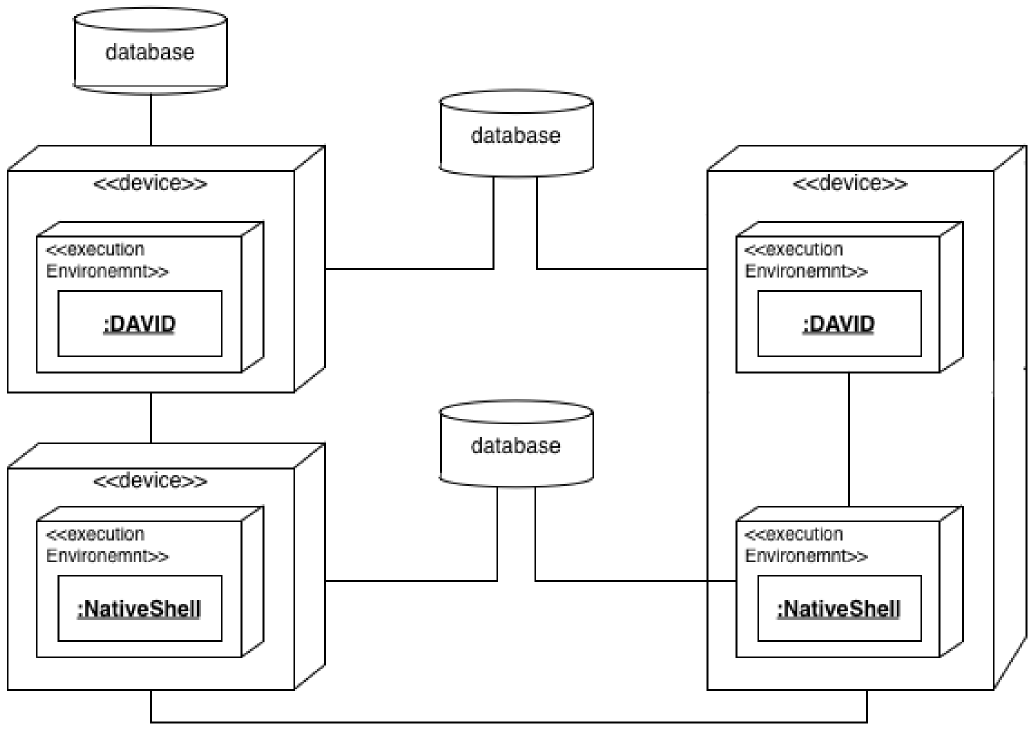


Figure 27: Deployment of DAVID and its native shell

development. As Martin [53] puts it: "At first it isn't so bad. An ugly wart here, a clumsy hack there, but the beauty of the design still shows through. Yet, over time as the rotting continues, the ugly festering sores and boils accumulate until they dominate the design of the application". Given the pervasiveness of the "rotting" phenomenon, software engineers have come up with countless ideas and concepts to prevent it. One of these ideas is the "separation of concerns" (SoC). SoC is "a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern" [89]. Even though many architectures support and encourage such separations, this does not necessarily mean that a developer will follow it. I enforce SoC by using two distinct programming languages for two different concerns. This separation is analogous to separating the program architecture into distinct packages (see Figure 22). The "Interaction" package is written in pure JavaScript, while the logic and data packages are written in Dart. The reasoning behind this strict technical separation is to force developers to choose either the "front end only" language of JavaScript, or the language that drives the logic of the system. Even though the languages are compatible and theoretically could influence packages they are not designed to influence, it is quiet difficult to mix both languages and still follow their control flow. In essence, it forces the developer to keep parts of the system which are responsible for the interaction in the "interaction" package. Parts that are designed to handle logic in the corresponding logic packages are written separately in Dart.

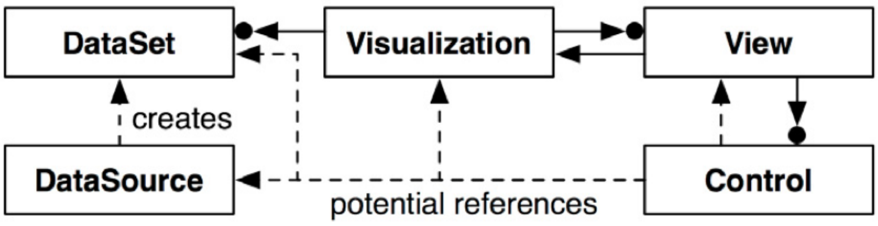


Figure 28: Reference model for data visualizations from [38]

6.2 ANATOMY OF COMPONENTS

The heart of DAVID is its flexible visualization components. Components consist of five basic classes: DataSet, DataSource, Visualization, View, and Control. These basic classes are defined in [38]s reference model of data visualizations and are shown in Figure 28. DataSets are instances of data that are brought into existence by a DataSource: a class that knows how to fetch data. A visualization builds a visualization based on a DataSet. The Visualization is managed by a View that represents the general-purpose interaction class for the user. Lastly, the Control manages the logic behind some, but not necessarily all, of those classes. Especially the visualization itself is not managed by the controller, but by the ViewPoint class. This is an unusual decision, especially in the context of the MVC architectural pattern that DAVID builds on. It allows the user to actively work with the code that is strictly responsible for the visualization of data, rather than data processing etc. Even though the authors of [38] clarify the association between Control and Visualization as a "potential" reference, one might expect the Control to provide the logic for all visual elements. However, this does not need to be the case.

DAVID does not define such an association. Visualizations have their own backing control code that can be modified during runtime. Figure 29 depicts how the reference model is applied in DAVID. These classes were introduced during the description of the architecture (see 6.1). Some of the associations between the classes of the reference model are realized by association classes (such as ViewPointValues, see 24).

The "Visualization" class is the center of the reference model. However, DAVID does not reduce the complexity of visualizing data into a single class.

Visualizations rely on three more classes (as mentioned in 6.1): ViewPoints, ViewPoint-Values, and ViewPointOverlays, which are instantiated for every visualization. Their concrete implementation is chosen during runtime by the "ViewPointComponent". This component is used by the orchestrating framework to hand over control flow to the extension points. ViewPointComponents use the factory pattern to create their corresponding accompanying classes.

Figure 30 depicts two examples of components. The framework plays the role of the client of the abstract factory and chooses a specific implementation of the MapView-Component. This class then decides which implementation of ViewPoint, ViewPointValues, and ViewPointOverlay are needed. However, the ViewPointComponent is different from the AbstractFactory pattern, as it saves those connections into the ViewPointValues, rather than the client of the factory, since the client will not have any control over those components after instantiation. This is a restriction originating from angular.Dart. Upon

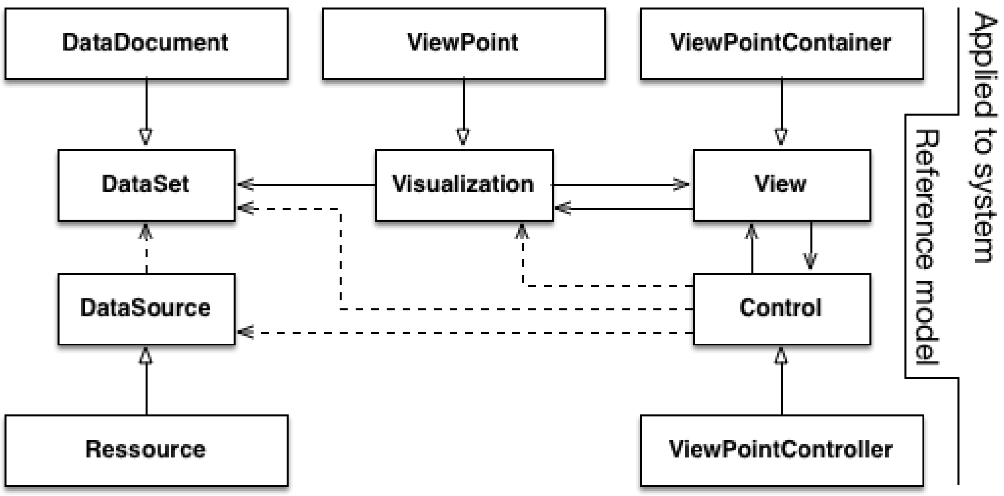


Figure 29: Implementation of the reference model

instantiation, the ViewPoint takes control and manages the visual representation of the data that is pushed in by the Controller.

6.2.1 *Working with pushed data*

Since DAVID is designed to work with massive amounts of data, it is crucial that the data not be duplicated (which may compromise performance), especially when two independent components visualize the same data set. This is why data passing is based on the Pipe-and-Filter pattern (see 16). Pushing data does not mean that data is copied and destroyed on multiple occasions, but rather, that a reference of the data is passed to various components. Filters can be applied on multiple occasions. The first instance of a filter is the Resource, which defines how data is fetched. After the data is returned, or even while it returns, the Resource can alter parts of the data. Though it is possible to alter data in the Resource, this should be avoided, as Resources have no knowledge of the data's context. A Resource merely marshals data. Marshaling means "transforming the memory representation of an object to a data format suitable for storage or transmission, and it is typically used when data must be moved between different parts of a computer program"[83].

6.2.2 *A data metric library*

Once data arrives at a ViewPoint, it needs to be prepared for the actual visualization (such as a line chart). Visualizations may have different requirements regarding the formatting or organization of data. As an example, consider a line chart and a bubble chart. Line charts are two dimensional and therefore need to know where a value falls on two axes. A bubble chart is three dimensional, since it encodes the size of the bubble in the chart, which represents one more axis. These examples show that although DAVID can have generic components, the behavior and, especially, how data is handled, need to be extremely flexible. Depending on the visualization's source data repository, precalcula-

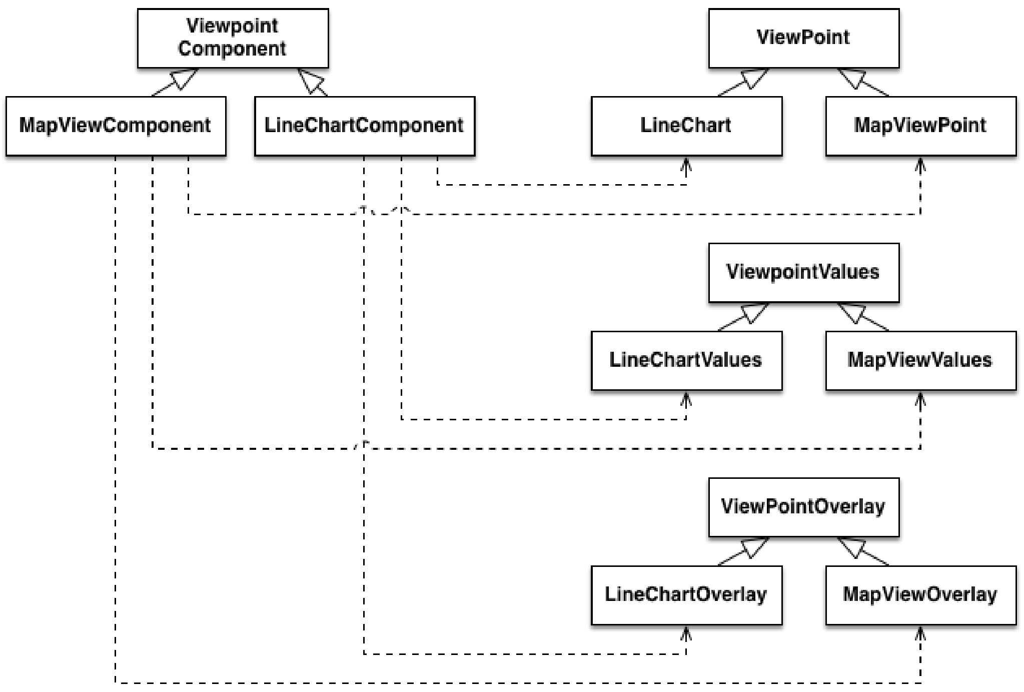


Figure 30: Abstract factory-based creation of ViewPoints

tion or aggregation might be necessary. DAVID uses libraries of metrics to address this issue. These metrics are preselected for each dataset and will be used for

- Preparation of the complete dataset
- Selection of a single element based on a single dimension of the data

In 6.3, I describe how these metrics are selected. Metrics are supplied on the basis of two methods of the ViewPoint class, “setData(data)” and “getDataPoint(dimension). Each of these methods invokes a corresponding library function capable of fulfilling the request. These metrics can also be altered during runtime, thus allowing for scripting and true flexibility in data preparation.

Dynamic data manipulation

Figure 31 depicts the flow and calls on the data, once it has arrived in a component. The “setData” method provides the entry method for the data. Once called, the system checks either for a user- or system-supplied “dataManipulator”. A data manipulator is responsible for executing lightweight computations on the whole dataset before it is used in the visualization process. It follows the ideas proposed in [28], in which filters are applied to data that leaves a DataSource. The data manipulator could also serve as a sampling algorithm that selects subsets of data on the fly (see [51]). Data manipulators are simple functions that can be overwritten by the user for each component; not only for each DataSource, but for every ViewPoint in a DataSource. The prepared data is then passed to a loop that looks at each of the elements (rows). For each element, the loop applies the “dataMetric”. This function is responsible for inserting the data into a

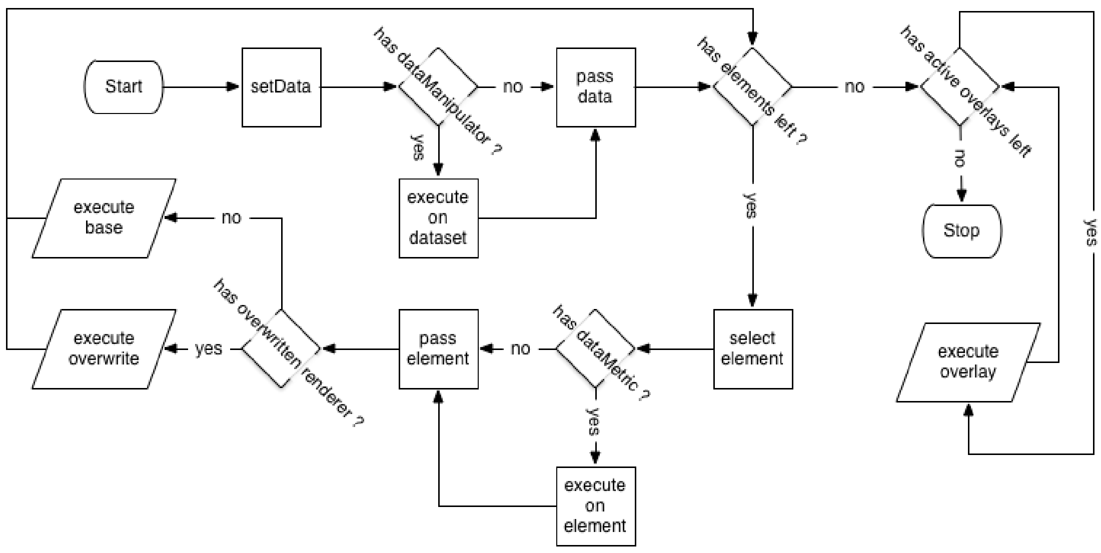


Figure 31: Data flow in a ViewPoint

visualization. It provides a mapping from an arbitrary input value to an arbitrary output value, allowing the user to associate an explorable dimension, such as time, with a value, such as heart rate (see the basic function in 5.3.1). The dataMetric function takes the current component's context and a value, and may be different for each dataset and each visualization. This also means that the dataMetric function is needed to add a new metric to each visualization (or refer to it) when extending the system. Each component has a rendering method, and when the metric is applied, the system checks for the custom rendering process. Listing 1 shows the interfaces available to each component. In addition to the variables that cache the metric, each component has an "onRender" method. If this default method is overwritten, the overwritten code is executed. This allows the user to alter the rendering on the fly.

The data access methods can be called with the help of wrapper functions (getDataPoint as a shortcut for the metric and getTimePoint for the reverseMetric). Note that the data metric is applied to all dataPoints. DAVID does not use an observer to watch for changes in the data; it only redraws data when the user actively commands the system to do so. When all elements have been rendered, the state of the overlay functions is checked. If a function is marked as active, it is applied as well. Listing 1 describes the variable "overlayLibrary". This variable holds a reference to an object that manages the defined overlays for the current component.

Listing 2 shows the functions that are available for the purpose of interacting with an overlay. Note that the function "toggleOverlay" asks for an overlay name, which in turn refers to the properties of the overlay library. Overlays are not stored in an array, but instead, the properties of the library itself are the possible overlays. These properties are objects themselves and manage their own encapsulated state. This unusual setup allows the user to interact with the overlays directly and easily, since they can be accessed by name during runtime.

Listing 1: Methods essential to the display and updating of a component

```
/*  Data functions
    These variables alter and select data based on the dataset and visualization */
dataMetric
reverseDataMetric
dataManipulator

/*  Lifecycle calls
    These methods are used during the lifetime of a component */
manipulateData(data)        // executes the dataManipulator
getDataPoint(time)          // executes the dataMetric to get values
getTimePoint(data)          // executes the reverseDataMetric to create
                            //reverse mapping between value and time
setContainer(container)     // sets the DOM container of the component
setData(data)               // called when data is pushed in
onRender(context)           // called whenever the data functions change
onTick(type,value, meta)    // when events come on the tick method is
                            // executed with all meta-information needed
```

Listing 2: Basic overlay interface

```
/*  Overlay functions
    These functions initiate and execute the overlays on a component */
executeOverlays
loadOverlays
toggleOverlay(name)

/*  Overlay behaviour objects
    These objects represent a single overlay each.  */
onSelectArea
onMaxMin
onAverage
...
```

6.2.3 *Event based communication*

Chapter 5.3 described how a user links components on a visual level. Internally, linking means that the system sets up a ViewPointConnector with a source and destination (see figure 24). ViewPointConnectors do not pass data from one component to another (with few exceptions), a constraint that implements the idea of the "Observer pattern". The purpose of this pattern is to allow program elements to register themselves for notifications about state changes for an object [36]. Different proposals have been made as to how such notifications should be transmitted, and if they should carry data (push) or not (pull).

In DAVID, components rely on pushing events. These events are used for reasons beyond simple communication between components. The ViewPointController talks to its corresponding ViewPoint using these events as well. This allows the view to be completely separated and independent from the controller (and vice versa), since no direct calls are made.

Following this logic, the specific sender of an event does not matter. This is how DAVID achieves its modular and flexible component communication. Figure 3 shows a list of possible events that DAVID uses to transmit information between components. The "TICK" events notify a ViewPoint that the current time variable has changed, whereas the "ANNOTATION" events signal that an annotation was either added or removed. The "PAUSE/RESUME" events describe whether time is moving, and the "OVERLAY" event signals that an overlay was added, removed, or altered. Once received, all events are pushed to the registered observers of the ViewPoint via the ViewPointConnector.

The links between components are characterized by their type. When pushing data to another component, the data is altered based on the type of connection (see 5.3.1 for a detailed description of the link types).

Listing 3: DAVID's known events

```
static const int TICK_TIME_EVENT = 0;
static const int TICK_TIME_MANUAL_EVENT = 1;
static const int ANNOTATION_ADD_EVENT = 2;
static const int ANNOTATION_DELETE_EVENT = 3;
static const int PAUSE_EVENT = 4;
static const int RESUME_EVENT = 5;
static const int OVERLAY_EVENT = 6;
static const int PRESENTATION_EVENT = 7;
```

6.2.4 *Representation*

As proposed in 5.2.4, the visual representation of a component relies on cascading style sheets (CSS). In [77] CSS is described as "a style sheet language used for describing the look and formatting of a document written in a markup language". CSS is a powerful way of describing the visual representation of elements in a DOM-based document, though it is not without problems. CSS relies on selector descriptions in the DOM, which have a global scope. Thus, CSS styling rulings cannot be selectively applied to a subset of elements in the DOM, and must instead be applied to the whole DOM. The only way to scope CSS is to use so-called "Web components", which create a hidden DOM tree within a DOM.

This hidden DOM is called the ShadowDOM[2]. The specification defines it as "a method of establishing and maintaining functional boundaries between DOM trees and how these trees interact with each other within a document, thus enabling better functional encapsulation within the DOM" [71]. Scoping the DOM on a "per-component" basis has a variety of positive aspects. As described in 6.2, it allows us to implement the component based concept into the DOM and supports the separation of concerns. More importantly, the ShadowDOM allows the system to apply CSS and the scripting mechanisms described in 6.2.2 and 6.4.1 on a "per-component" scope. This allows the user to use the same selectors in different components with different semantics. This is particularly helpful for general selectors such as:

2 Standard proposal: http://w3c.github.io/webcomponents/spec/shadow/

```
.line{...}           // Describes the styling of a line in a line chart
.dot{...}            // The dots in the line of the line chart
.dotInfo:hover{...}  // The style of the dot if hovered
```

DAVID makes heavy use of the ShadowDOM and its closed scope. For example, DAVID does not allow components to see the inner contents (the structure) of other components without specifically establishing a connection between components. Once established, components can influence each other programmatically. This approach is unusual for web- and DOM-based applications since it forces developers to carefully set up connections between certain program parts, though it is increasing in popularity.

6.3 ANATOMY OF A DATA DOCUMENT

All data in DAVID is accessed through a DataDocument. DataDocuments are basically self-contained MVC models. They are capable of fetching data, preselecting and filtering data, and offering access to this data. They also manage the creation of certain context elements, such as metrics and annotation libraries, and store meta-information during runtime. DataDocuments are thus the central class for working with data.

The ecosystem of a DataDocument is depicted in figure 32. Each DataDocument works with a Repository, which is a collection of data-calls. These data-calls rely on so-called "resources". A concrete implementation of a resource offers a mechanism to fetch data, but it knows nothing about the data itself. A resource could also be conceptualized as a driver. Examples of this type of resource include SQL Resources and File Resources. When a DataDocument requests data from a Repository, the Repository creates a RepositoryResult. These RepositoryResults follow the idea of a Future / Promise, as already discussed in 5.4.2. Although the Repository directly returns this result, it does not necessarily contain data. The Repository will populate the content of the Result once data arrives from the Resource. The Promise, in turn, only "promises" the DataDocument that at some point it will offer data. This allows the DataDocument to execute without interruption. The DataDocument can also define operations that execute once the RepositoryResult fulfills its promise (which must occur, even if no valid data are returned). The Repository also defines a Metriclibrary that is passed to the RepositoryResult. The metric library is highly context-specific. It depends on the data queried and is responsible for providing a library of functions that can operate on the data and prepare it for the visualization. However, this library is not actively used by any of those classes. Instead, the ViewPointValues class takes the library and passes it to the controller. Finally, RepositoryResults have a DataAnnotationLibrary that is also defined during the creation of the result, and defines with which markers a dataset can be marked. Once the RepositoryResult returns data, the user can use this library to mark certain areas of the data with annotations (a list of DataAnnotationItems). Note that these Annotations are all added by the ViewController.

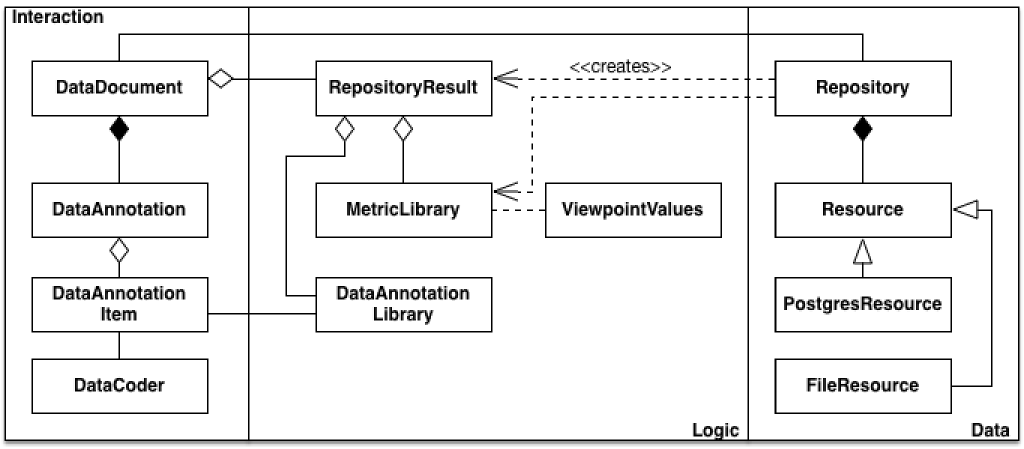


Figure 32: Context of a DataDocument

6.3.1 *Annotation and metric libraries*

A repository groups a set of related data calls and allows the user to explore this set of data calls. A concrete example of a Repository is the AgeLabRepository, which offers a set of calls that fetch data from the AgeLab Database server. There might also be a different repository, such as a repository that fetches data from Twitter. Figure 33 shows an example of such a setup. Each repository generates its own implementation of a DataAnnotationLibrary and a MetricLibrary. A user picks a repository, selects a data call, and the concrete Repository prepares the result for the DataDocument that will handle the data later on. All library implementations are context specific. This means that the libraries do not necessarily share behavior or structure, other than the calls that create them and retrieve their basic information. Metric libraries are intended to have different behavior and side-effects on the data for every Repository.

6.4 IMPLEMENTATION TECHNOLOGIES

This work builds on a large number of web-based technologies, which leaves the project vulnerable to problems stemming from the properties of web-based programming languages and frameworks. JavaScript, for example, runs in a runtime environment in the browser. How the language is interpreted depends on the browser that implements it (or

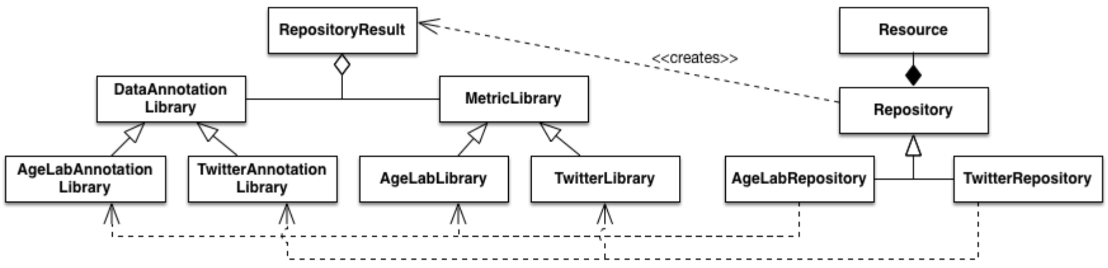


Figure 33: Dynamic creation of RepositoryResult

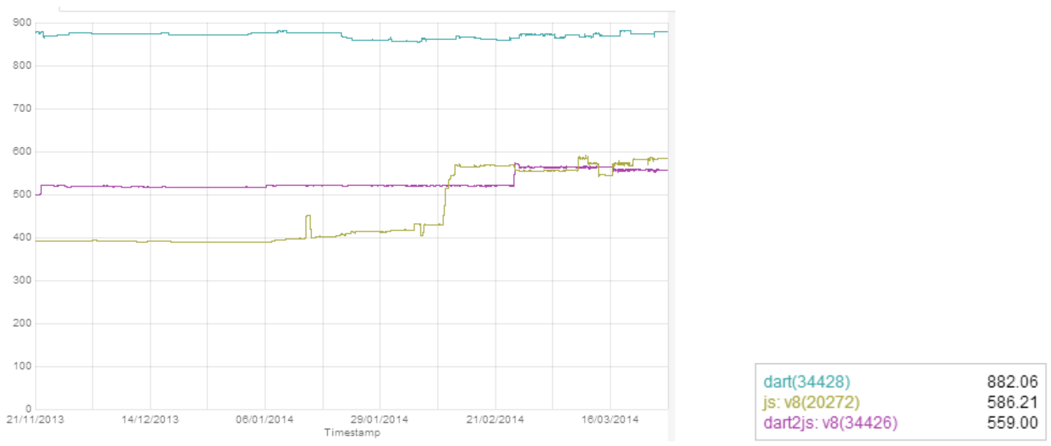


Figure 34: Speed metrics of Dart compared to JavaScript and the cross language compiler Dart2js (higher is better) [3]

its runtime engine). The biggest problem is that JavaScript does not have the properties of a real object oriented language, such as inheritance or a class system. This is highly problematic when designing larger applications, as conventional modeling techniques often fail and best practices for this language are still evolving. For this reason (and many other smaller problems with the language), numerous packages, additions, and frameworks for the language have evolved, aiming to fix and work around these issues. Prime among them is the standard on which the current JavaScript implementations are built: ECMAScript. However, the current ECMAScript version does not bring features such as classes and inheritance to JavaScript. Those features are part of the future milestones of ECMAScript 6 and 7, which are still far from release. This makes JavaScript a problematic choice for large applications.

Languages such as CoffeScript, Objective-J, and Typescript, which have evolved to fill the object orientation gap, rely on completely different paradigms. One of the newest players in this field is Dart, which is actively supported and developed by Google. Dart's single (and very ambitious) purpose is to replace JavaScript. Dart brings structure into development by introducing object-oriented techniques such as inheritance, classes, packages, and real libraries. Dart is extremely fast when it comes to certain basic operations, which is one of the main reasons why I chose to built this work on top of it. The data layer described in 6 may potentially work with millions of data points. In this context, the speed of basic operations is crucial. The Dart development team at Google focuses on this speed aspect in their runtime environment, which is included in the Chromium branch of the Chrome browser. Figure 34 depicts the speed of the Dart language in comparison to JavaScript (the yellow data line). Additionally, Dart provides a strong interaction layer to JavaScript. Calls can be made between the two languages, and Dart can also compile its own source-code on the fly to JavaScript, thus allowing it to run in a browser that does not support it. This makes it a viable option as a "structured" language, as one can fall back to JavaScript code if needed. this is necessary because this work builds on the extremely powerful libraries that have evolved in the ecosystem of JavaScript. Even though most of the parts of this system are written in Dart, the user is able and encouraged to interact with the JavaScript layer (as described in 6.1).

Conquering type uncertainty

Some of the concepts and mechanisms that have been introduced, such as metric libraries (see 6.2 and 5.4), produce heterogeneous data. The types defining these data are only known to the metrics or components which assume certain types. Therefore, the language used for implementation must be able to conquer this type uncertainty. Dart is a language with optional types, in that if the type is undefined it is inferred by the runtime system. This allows DAVID to work quite efficiently with its data, since types are not required but can be used once the knowledge about the data is obtained.

6.4.1 *Data driven documents*

As discussed in chapter 5.2.2, this system not only allows the user to work with the data, but actively encourages the user through library-agnostic implementations. The ability to choose a library that is best suited the target of a certain visualization (maybe during a presentation) is a freedom that most tools don't give their users. This flexibility comes with the duty to choose wisely among the enormous ecosystem of technologies available for visualization. Some widely used libraries are already part of DAVID. Figure 35 shows the logos of such libraries. The system initially supports "D3" and "Highcharts"[3]. This selection reflects the current most popular libraries for drawing and chart generation. D3, the "Data Driven Documents" library, has risen in popularity as it "allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document."[5]. D3 focuses on drawing interactive figures and giving programmatic access to the elements on the screen. D3 has a very active community, with 7,302 questions[4] on Stackoverflow[5] at the time of this writing.

Highcharts offers "intuitive, interactive charts to your web site or web application. Highcharts currently supports line, spline, area, areaspline, column, bar, pie, scatter, angular gauges, arearange, areasplinerange, columnrange, bubble, box plot, error bars, funnel, waterfall and polar chart types".[6]. It provides the user with a ready-to-use set of basic charts. Both libraries are used in the provided components (see 7.1). However, these libraries may not always be the best choice for a given visualization problem. For example, when drawing millions of points onto a canvas, a pure SVG library might be a better choice. To move beyond these two libraries, I suggest two more libraries for typical data visualization: Raphael and ProcessingJS.

PROCESSINGJS [6] is based on the well-known "Processing 2.0" environment [7]. It is the corresponding library fork for the JavaScript language and features a wide set of library functions that incorporate videos, 3D rendering, mathematical functions, and even camera and lightning systems. This library is particularly helpful when drawing huge interactive visualizations. Processing is a DSL, since it uses specific terms such as "circle" and "rectangle" to hide abstractions. Raphael and D3 take a different approach.

3 Product website: http://www.highcharts.com/
4 Question archive: http://stackoverflow.com/questions/tagged/d3.js
5 Project website: http://stackoverflow.com/
6 Project website: http://processingjs.org/
7 Project website: http://processing.org/

Figure 35: Typical and powerful libraries and DSLs: D3[5], Raphael [12], Processing [9], Highcharts [6] (left to right)

RAPHAEL is one of the many libraries that compete with D3. Raphael focuses on direct access to an SVG layer and allows vector based manipulation of the graphics. It differs from D3 in that it allows free drawing of pixel-based objects, and does not allow data binding. Because Raphael's objects are unbound, direct manipulation of objects on the screen is more straightforward, even though it is not as convenient when data must be handled consistently.

Both libraries enjoy massive community support. See 6.2 for guidelines on how these libraries can included in DAVID.As discussed on multiple occasions, it can be problematic to force a DSL onto the user. This is why all of libraries are imported and supported. A user can choose freely between them. Due to the flexibility and the closed scope (see 6.2, 5.2.2 and 5.2.4) of components, they can even be used in conjunction.

7 EVALUATION

DAVID relies not only on its visualization components. The visualization components can only be used if there is a suitable frontend to create the dashboards in which those visualizations live. This chapter showcases such a frontend that is part of DAVIDs reference implementation.

Figure 36 shows how a user selects a repository. On the top, all available repositories are shown from which the user can choose. Once selected a list of dynamically loaded data-calls is presented (the black box). Such data-calls can then be configured if needed. The list of parameters for the data-calls is not a generic interface but created on a "per-data-call" basis.

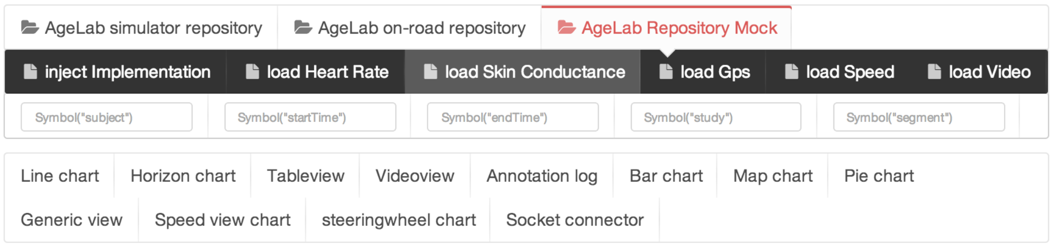
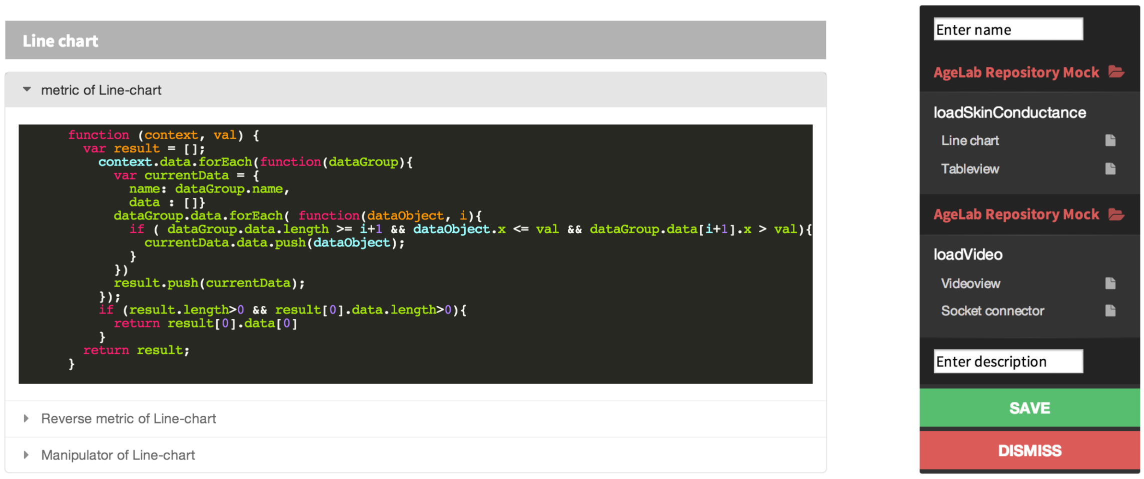


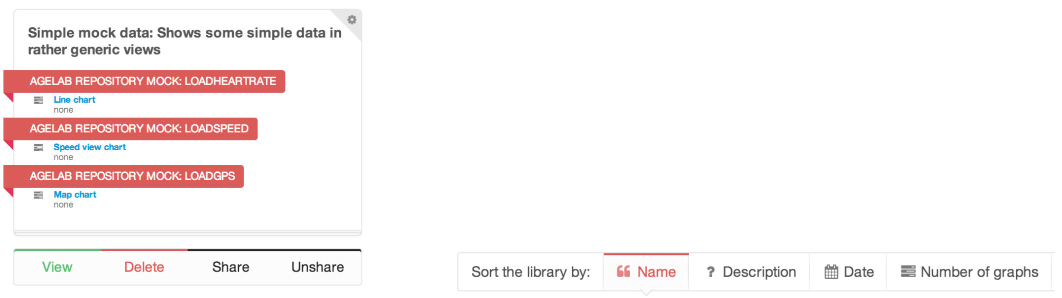
Figure 36: Selection of Repositories

After configuring the data-call, the user can select the ViewPoint he wants to use for the data. Multiple ViewPoints, as described in 6 and 5 can be used for the same dataset. Figure 37 depicts how a user proceeds after selecting a ViewPoint. Each ViewPoint can be customized. A "Metric", "Reverse metric" and "Manipulator" can be defined. Those elements are the functions for translating data into a format a ViewPoint can work with (see 5.3.1).



```
function (context, val) {
  var result = [];
    context.data.forEach(function(dataGroup){
      var currentData = {
        name: dataGroup.name,
        data : []}
      dataGroup.data.forEach( function(dataObject, i){
        if ( dataGroup.data.length >= i+1 && dataObject.x <= val && dataGroup.data[i+1].x > val){
          currentData.data.push(dataObject);
        }
      })
      result.push(currentData);
    });
    if (result.length>0 && result[0].data.length>0){
      return result[0].data[0]
    }
  return result;
}
```

Figure 37: Data metrics (left) and creation of the DataDocument (right)


Simple mock data: Shows some simple data in rather generic views

AGELAB REPOSITORY MOCK: LOADHEARTRATE
Line chart
none

AGELAB REPOSITORY MOCK: LOADSPEED
Speed view chart
none

AGELAB REPOSITORY MOCK: LOADGPS
Map chart
none

Figure 38: DataView selection (left) and ordering (right)

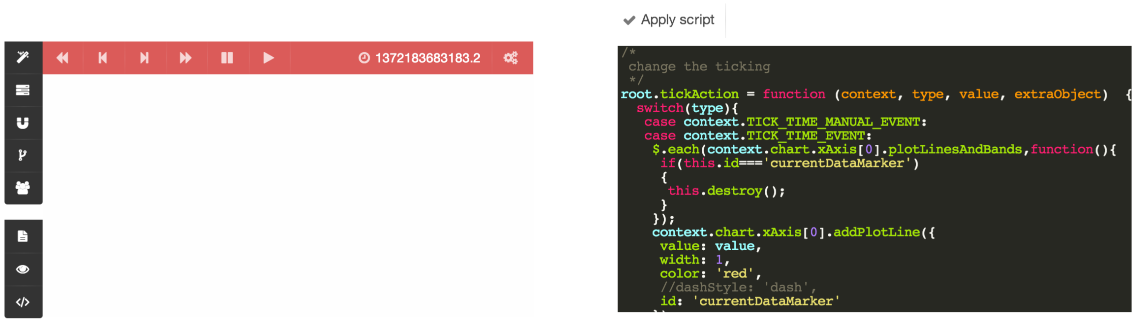
These fields are pre populated and work in a predefined yet context specific way. If a user wants to use and save the selection of Repositories, her or she can add it to the DataDocument. DataDocuments are collected on a stack and saved later on. Figure 37 shows on the right side such a stack. DataDocuments consist of a Repository (red) , data-calls (white) and ViewPoints (indented from the data-call).When a user finished the creation of a collection of DataDocuments they are saved and grouped into a DataView. Those DataViews are then presented as a card to the user. Figure 38 shows such a card. DataDocuments are again depicted as a red box and the corresponding Repository and data call. All ViewPoints are placed beneath this. Since a user might have a variety of DataViews he needs to be able to search and explore his library of DataViews. Selecting one of the ordering buttons presents the user with a list that is ordered by the selector.

7.1 THE VIEWPOINT

The concepts that I described in 6 and 5 have all been implemented. Each ViewPoint has a container that exposes some reoccurring elements. This allows the user to build and reuse a mental model he learns when using DAVID for the first time [8]. Each ViewPoint is surrounded by an action-bar and a controller-bar as shown in figure 39. The action-bar (left) allows the user to select operations in the ViewPoint.

From top to bottom the buttons allow

- Opening the overlay activation menu (see figure 40)
- Showing data context (see 40)

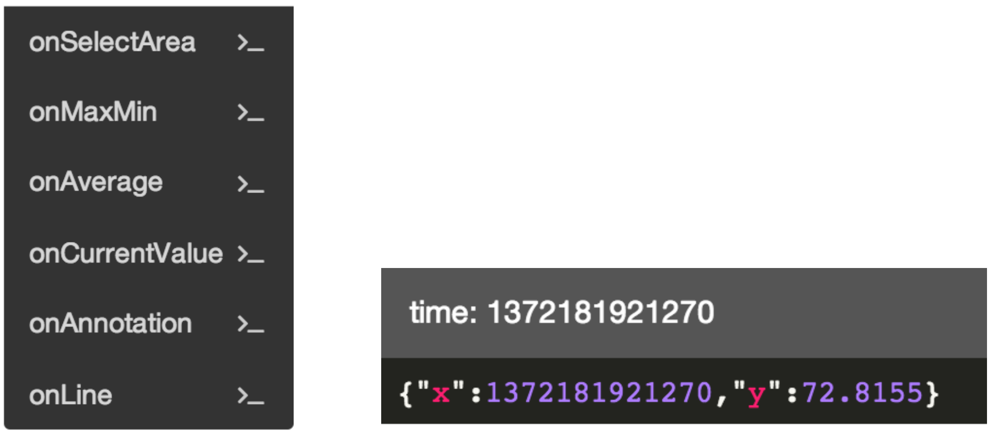


```
/*
 change the ticking
 */
root.tickAction = function (context, type, value, extraObject)  {
  switch(type){
   case context.TICK_TIME_MANUAL_EVENT:
   case context.TICK_TIME_EVENT:
    $.each(context.chart.xAxis[0].plotLinesAndBands,function(){
     if(this.id==='currentDataMarker')
     {
      this.destroy();
     }
    });
    context.chart.xAxis[0].addPlotLine({
     value: value,
     width: 1,
     color: 'red',
     //dashStyle: 'dash',
     id: 'currentDataMarker'
```

Figure 39: A ViewPoint (left) and its backing script (right)

- Allowing linking (see 41)
- Exposing linkage type menu (see 41)
- Marking other ViewPoints with the same backing DataDocument
- Showing the annotation menu
- Showing of visual style definition (same as 39
- Opening the scripting menu (see 39

The controller-bar gives the user control over movement in time. There are a variety of ways to manipulate the time once it has started ticking. One of the key aspects of DAVID is to allow the user to interact with the ViewPoint. Figure 39 shows on the right side the scripting window a user can access and use to alter behavior. This window does not only allow to change runtime behavior but even the provided metrics can be access and altered here.



```
time: 1372181921270
{"x":1372181921270,"y":72.8155}
```

Figure 40: Overlay activation menu (left) and ViewPoint data context (right)

Additionally the user might execute the provided overlays. Those context specific functions are exposed in the menu, shown in figure 40 on the left side. Selecting one of those functions executes the predefined or overwritten code. All of those overlays have access to the main window of the ViewPoint but also to an area that shows context information. This is particularly helpful if a visualization interpolates over values or reduces the dimension of a dataset. Those information can then be shown there.

To link a ViewPoint to another, the user only needs to draw a line from the marked areas of the ViewPoint to another ViewPoint. The connection automatically snaps at its correct place once the mouse is released. Figure 41 shows on the left side such a connection between two ViewPoints. Once connected, the type of this connection can be altered through the connection-type menu (right).

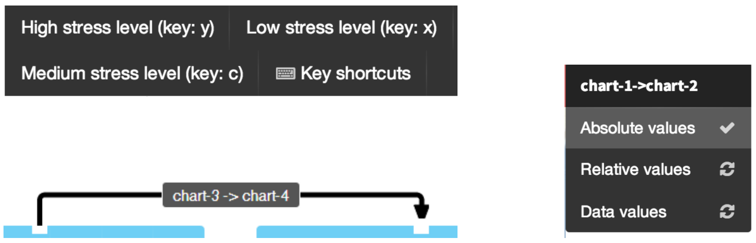


Figure 41: Annotation buttons (top left), linked ViewPoints (bottom left) and type of linkage (right)

Besides the already introduced "time" (here named "absolute values") and value based linking (here named "data values") the user can also select "relative" values which means that time is ticking relative to its starting point but with synchronized speed and step-width.

All backed data can be annotated (as designed in 6.3). The annotation-bar shows the content of the available DataAnnotationLibrary. Figure 41 shows this bar. By clicking one of those annotation buttons an annotation is added at the current time position to the original dataset. User can also accelerate in their usage by using predefined shortcuts, once they know the system . Lettings users accelerate is crucial to let them become experts [18].

A completely opened ViewPoint (except the code windows) in its container is depicted in figure 42 as a reference.

On multiple occasions it was stated that the real value of DAVID lies not only in its singular components but in the possibility to connect and link data. Since DAVID is a visual system a screenshot of a complete and running data view shall not be withheld. Figure 43 shows a configuration that visualizes a subject that was driving in Boston, MA.

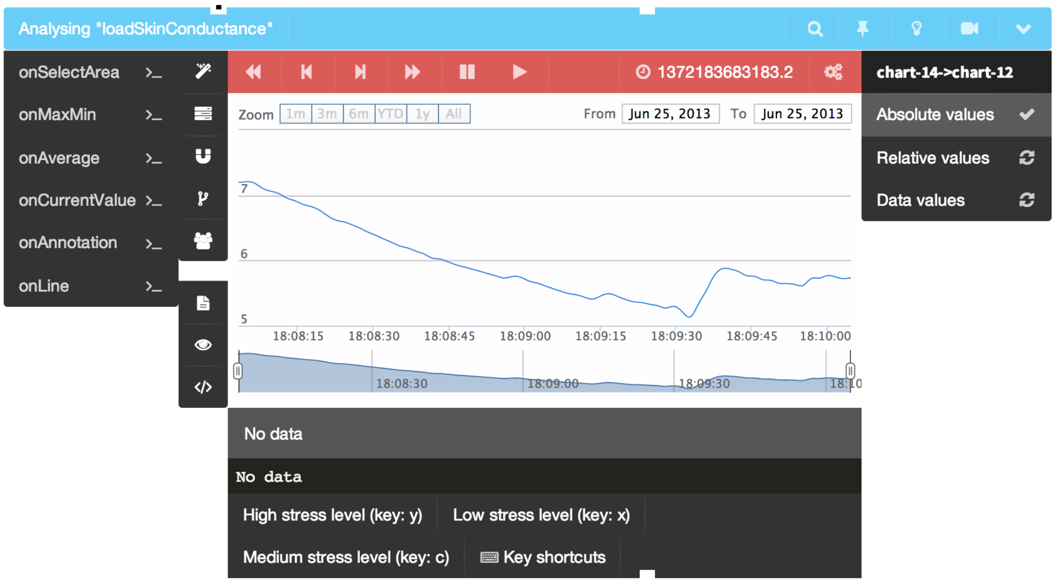
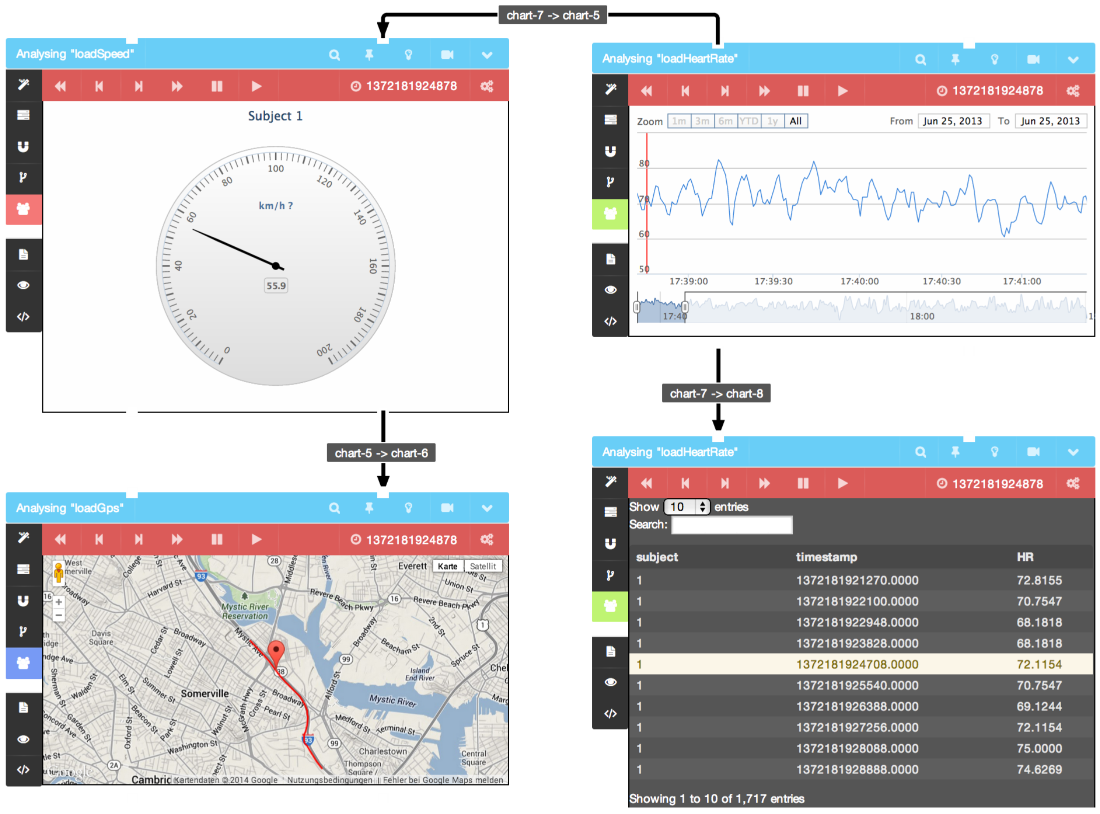


Figure 42: A complete LineChart ViewPoint



Show 10 entries
Search:

subject	timestamp	HR
1	1372181921270.0000	72.8155
1	1372181922100.0000	70.7547
1	1372181922948.0000	68.1818
1	1372181923828.0000	68.1818
1	1372181924708.0000	72.1154
1	1372181925540.0000	70.7547
1	1372181926388.0000	69.1244
1	1372181927256.0000	72.1154
1	1372181928088.0000	75.0000
1	1372181928888.0000	74.6269

Showing 1 to 10 of 1,717 entries

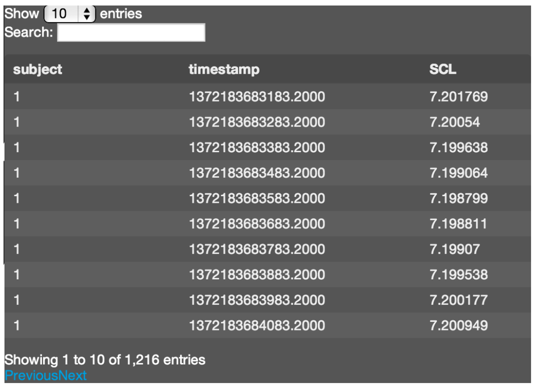
Figure 43: Complete DataView that shows a driving subject

On the upper right the heat rate is monitored. As time moves a red line marks the current value. A Speedometer ViewPoint and a table view is connected to this ViewPoint. The colors on the left corner of each ViewPoint show if a ViewPoint is backed by the same data as another (same color). The speedometer is then linked to the GPS data which is shown on the MapView ViewPoint.

Implemented ViewPoints

To showcase the power of DAVID, a variety of ViewPoints have been implemented. Some of which have not only visual behavior but allow to annotate and share data. Figure 44 shows the table Viewpoint and the LineChart ViewPoint. Tables simple show the complete backing data in chunks. When "playing" the data the currently active row gets highlighted as shown in figure 43. Additionally the user can move between pages by hand (on the bottom left) or even execute a full text search on the data). Line charts plot time on the x- and the corresponding value on the y axis. Those ViewPoints also allow to select time intervals either graphically (on the bottom) or by data (top right). Line charts also allow direct interaction such as moving the mouse over the data and seeing the exact values in a context menu. Other manipulations offered by the overlay activation menu (see figure 40) include changing the line to distinct points and jumping in time by click.

Figure 45 showcases the MapView and the SpeedView. The MapView relies on Google maps and draws the complete dataset, interpreted as GPS positions on the map. A red



Show 10 entries
Search:

subject	timestamp	SCL
1	1372183683183.2000	7.201769
1	1372183683283.2000	7.20054
1	1372183683383.2000	7.199638
1	1372183683483.2000	7.199064
1	1372183683583.2000	7.198799
1	1372183683683.2000	7.198811
1	1372183683783.2000	7.19907
1	1372183683883.2000	7.199538
1	1372183683983.2000	7.200177
1	1372183684083.2000	7.200949

Showing 1 to 10 of 1,216 entries
PreviousNext

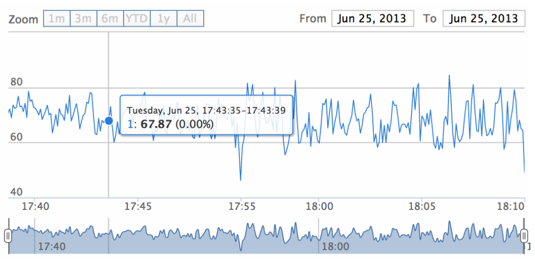


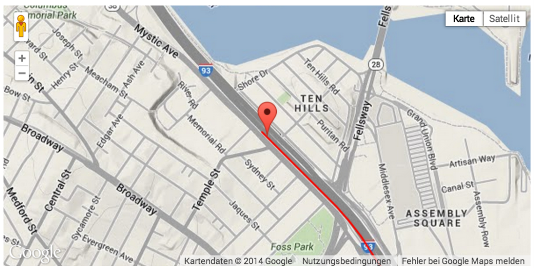
Figure 44: Table ViewPoint based on data tables (left) and LineChart based on Highcharts (right)

marker marks the position of the vehicle at the current time. The complete Google maps API can be used, thus allowing to use street view when analyzing data. This way a data-analyst can examine the surroundings of the car. The SpeedView shows the current speed of the car. The needle, showing the current value moves when the data is played.

Since DAVID is developed with the MIT AgeLab, some ViewPoints focus on specific data analysis problems of this lab. One of them is the angle of the steering wheel. Since subjects drive on the highway, the steering wheel is mostly only turned by a few angles. Figure 46 depicts on the left side a visualization of the steering wheel. The green dot is the center, the blue dot marks the actual position of the steering wheel. If the blue dot is in the middle of the surrounding circle the car is driving straight. Since the angle only changes very gradually a red dot magnifies this effect by multiplying the angle with 10. This way even small changes in the angle can be seen. A VideoView is available to watch the footage taken while driving. Moving quickly through the video is possible by the timing controls. When linked to a map view a video can even be viewed on the basis of GPS coordinates.

Classical bar-charts to compare values between different datasets can also be used to look at data in DAVID. Bars are generated on a "per dataset" basis, in this case for each subject. This chart proves quiet helpful when one wants to compare multiple datasets at a given point in time, since the bars only show the value at a point x in time.

Additional to the visualizations, two components have been implemented that allow more than just viewing data. Figure 47 shows a NetworkConnector and a Annotation-



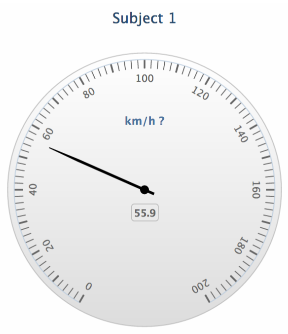
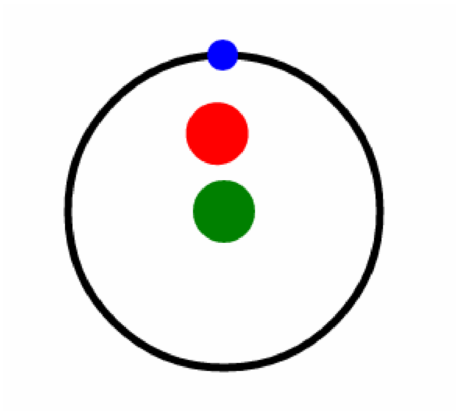


Figure 45: MapView based on Google maps (left) and SpeedView based on Highcharts (right)



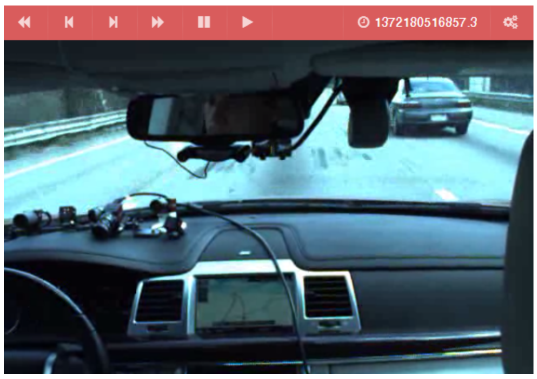


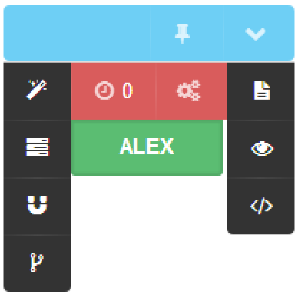
Figure 46: SteeringWheelChart based on D3 (left) and VideoView based on popcorn.js (right)

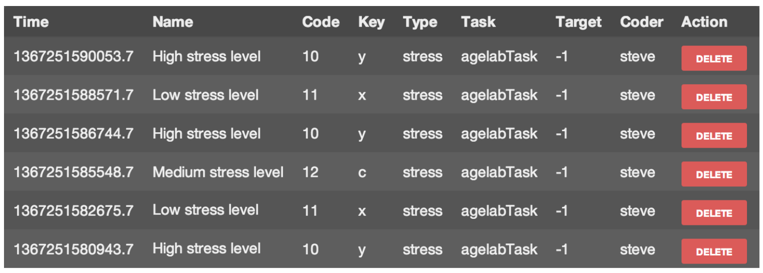
View. The NetworkConnector can connected separate instances of the application. The image depicts a connector that found one person in the network it can send data to. Once connected to a visualization this visualization can send or receive events to link and synchronize visualizations on completely separate machines. Only the selected machine (shown by a green button) receives those events. This allows a data analyst to rapidly build horizontally distributed data visualizations if one machine is not powerful enough. On the right side of the figure an AnnotationView shows how annotated data looks. Once this view is connected to a ViewPoint and the corresponding annotation is selected this table aggregates all those annotations. When the user is finished with his annotations, he can export them into a CSV format.

7.2 REPOSITORIES

To provide an effective prototype DAVID needs to be able to supply a set of repositories a data analysts at the AgeLab can use to experience the system. To introduce DAVID slowly into the workflow these repositories feature basic queries at the time of this writing but are supposed to grow over time. DAVID offers three repositories to the user.

THE ON-ROAD REPOSITORY offers access to data collected in real cars while subjects where driving. However, not all possible configurations one can create for the data-calls result in meaningful data. Some of the datasets are so old that the current queries return inconsistent data.





Time	Name	Code	Key	Type	Task	Target	Coder	Action
1367251590053.7	High stress level	10	y	stress	agelabTask	-1	steve	DELETE
1367251588571.7	Low stress level	11	x	stress	agelabTask	-1	steve	DELETE
1367251586744.7	High stress level	10	y	stress	agelabTask	-1	steve	DELETE
1367251585548.7	Medium stress level	12	c	stress	agelabTask	-1	steve	DELETE
1367251582675.7	Low stress level	11	x	stress	agelabTask	-1	steve	DELETE
1367251580943.7	High stress level	10	y	stress	agelabTask	-1	steve	DELETE

Figure 47: NetworkConnector (left) and AnnotationView (right)

THE SIMULATOR REPOSITORY exposes data-calls to studies that have been done in the in-house simulator. They are in most ways syntactically equal to the on-road datasets. GPS data however is missing.

A MOCK REPOSITORY offers a small set of sample data. This set of data was used to test all ViewPoints and is the best possibility to play with the system without requesting data. This repository exists only for training and demonstration. The word "mock" in the repository name refers to the concept of a mock object which mimics other objects.

These repositories do not all share the same data-calls. Figure 4 shows a list of possible calls a user can access from these repositories in the user-interface shown in figure 36.

Listing 4: A selection of implemented data-calls

```
loadAvailableSubjects
loadHeartRate
loadSkinConductance
loadNbackResponseTimes
loadGpsPosition
loadAcceleration
loadSpeedOfCar
loadSteeringWheelAngle
loadVideo
```

7.3 EVALUATION OF VISIONARY SCENARIOS

In chapter 2.5, three visionary scenarios were showcased. To measure the outcome these scenarios were used and compared to the as-is scenario. DAVID offers a UI to create the workspaces which are referenced in the scenarios. To compare the prototype for each scenario one workspace was created. Each workspace used real data from the AgeLab Repository (see 7.2). The following screenshots show how the situations described in the scenarios look in the prototype.

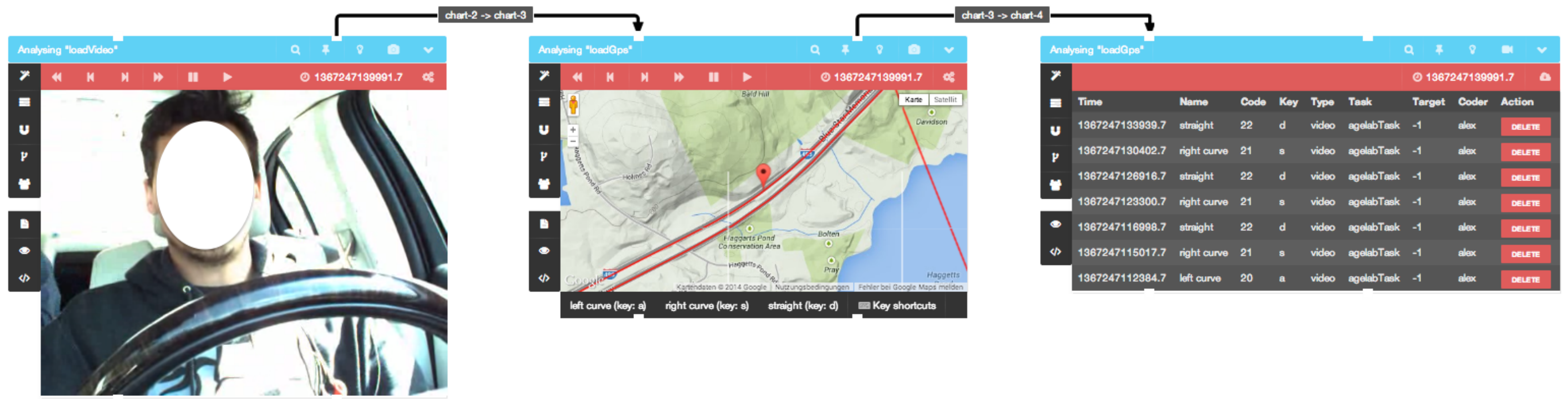
- Name: Annotate (see 2.5)

  Summary: See a set of data dimensions and annotate meta-information.

The "annotate" scenario requires three components: a video component showing the current video feed at point X in time, a map component showing the position of the car, and an annotation component, aggregating the annotations of the map. Such a setup is shown in figure 48. The video is linked to the GPS and the GPS to the annotation. The GPS data can be annotated with meta-information about the road. Additionally, the annotated data can be modified and exported after the session.

- Name: Patterns (see2.5)

  Summary: Compare two experiments and find unusual patterns



Time	Name	Code	Key	Type	Task	Target	Coder	Action
1367247133939.7	straight	22	d	video	agelabTask	-1	alex	DELETE
1367247130402.7	right curve	21	s	video	agelabTask	-1	alex	DELETE
1367247126916.7	straight	22	d	video	agelabTask	-1	alex	DELETE
1367247123300.7	right curve	21	s	video	agelabTask	-1	alex	DELETE
1367247116998.7	straight	22	d	video	agelabTask	-1	alex	DELETE
1367247115017.7	right curve	21	s	video	agelabTask	-1	alex	DELETE
1367247112384.7	left curve	20	a	video	agelabTask	-1	alex	DELETE

Figure 48: An implementation of the annotation scenario in DAVID

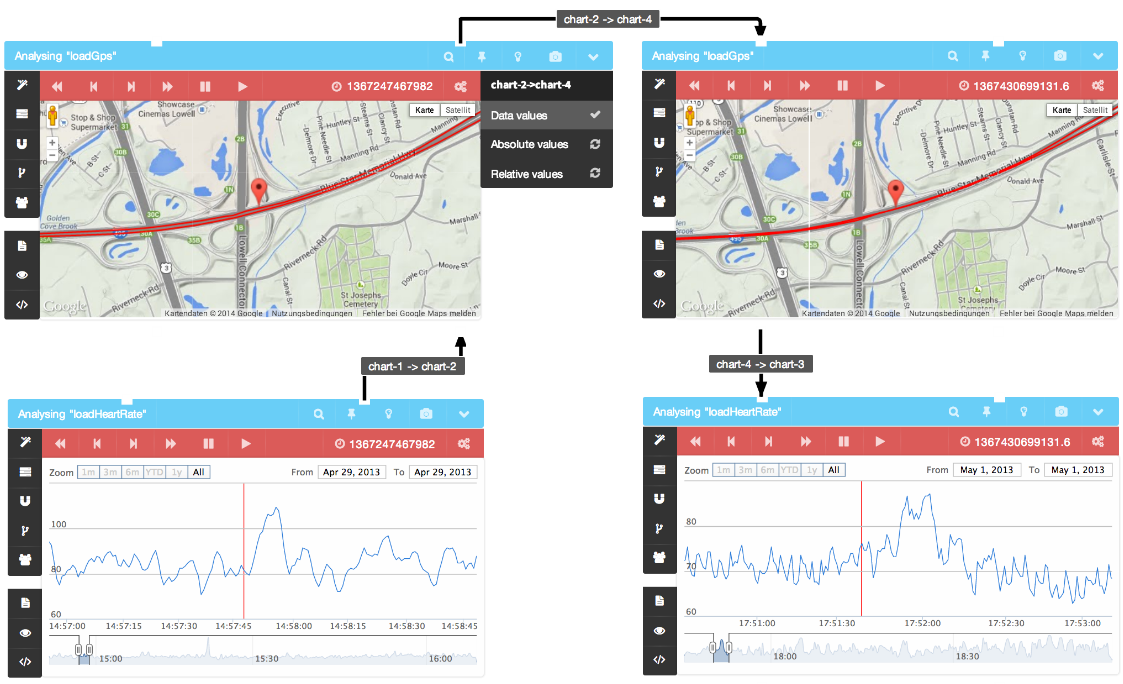


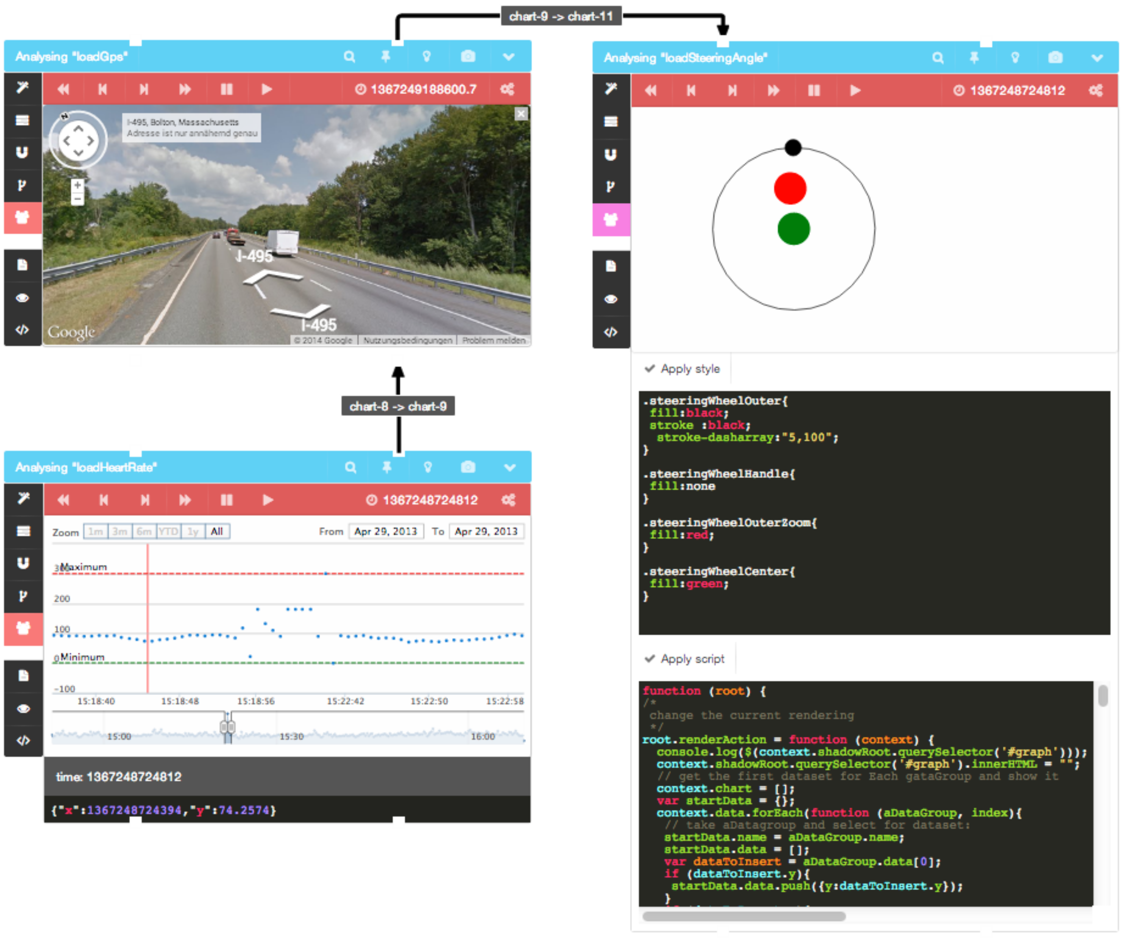
Figure 49: Comparing GPS coordinates in DAVID as described in the pattern scenario

In the “patterns” scenario the analyst is truly exploring the data. Figure 49 demonstrates this scenario. Looking at two different subjects the same pattern can be seen near an exit. The heart rate of subject 1 is connected to the GPS position. The GPS is then linked to the GPS of subject 2. The top-left window shows that the link uses GPS-Linking rather than Time-Linking (see 5.3.1). Therefore, both maps show the same GPS coordinate. The map of subject is then linked to the corresponding heart rate which shows a peek at the next exit.

- Name: Customization (see 2.5)

  Summary: Create a customized graph for an annotation session.

At the heart of the “customization” scenario lies the requirement for creative freedom (see FR 090 in chapter 2.4) and freedom of choice (see FR 080 in chapter 2.4). Creative freedom refers to the ability to manipulate the inner behavior of the component and the drawing. Figure 50 showcases this scenario. On the bottom left a custom line-chart was modified to show points instead of using a line. Additionally, an overlay showing the maximum and minimum value has been activated. The data of this plot is linked to the map component in the top left showing a photo of the street rather than the 2d map-representation. The visualization of the steering wheel is linked to this position on the street. On the right in figure 50, the code and style window for the steering wheel have been opened and used to create a completely new and unforeseen visualization that magnifies the degree the steering wheel has turned (top-right).



```
{"x":1367248724394,"y":74.2574}
```

```
.steeringWheelOuter{
 fill:black;
 stroke :black;
  stroke-dasharray:"5,100";
}

.steeringWheelHandle{
 fill:none
}

.steeringWheelOuterZoom{
 fill:red;
}

.steeringWheelCenter{
 fill:green;
}
```

```
function (root) {
/*
 change the current rendering
 */
root.renderAction = function (context) {
  console.log($(context.shadowRoot.querySelector('#graph')));
  context.shadowRoot.querySelector('#graph').innerHTML = "";
  // get the first dataset for Each gataGroup and show it
  context.chart = [];
  var startData = {};
  context.data.forEach(function (aDataGroup, index){
   // take aDatagroup and select for dataset:
   startData.name = aDataGroup.name;
   startData.data = [];
   var dataToInsert = aDataGroup.data[0];
   if (dataToInsert.y){
    startData.data.push({y:dataToInsert.y});
   }
```

Figure 50: Changing the behavior of components as described in the customization scenario

8 CONCLUSION

This work takes a new approach toward data-visualization systems. After describing common problems and challenges confronted by data-analysts, I analyzed the work already done in this field and highlighted current approaches in the field of data visualization. This analysis led to the proposal of a system that is able to dynamically and interactively visualize complex, linked data. The system is is designed with the needs of data analysts and the latest developments in the field of data visualization in mind. Typical problems that arise while visualizing data have been taken into account to outline a system that can be used by both novices and experts. In this proposal I focus on the system's user-friendliness, its ability to manipulate and dynamically visualize heterogeneous and linkable data, and on adding interactivity to such visualizations to allow for exploratory data analysis. I describe how creative freedom can be achieved through the use of such a system and outline strategies to reduce the complexities that big-data queries impose on a user. After describing the system conceptually and in terms of its inner software structure, I showcased my proposal by a proof-of-concept implementation. The developed prototype is actively used in data-analysis, visualization, and annotation at MITs AgeLab.

8.1 REQUIREMENTS EVALUATION

In chapter 2.4 a set of requirements was identified. As shown in the last chapter (7) many of those requirements have not only been considered in the design but have also been implemented. Table 4 shows the current status of the requirements. All important and high priority requirements were implemented. Before a requirement was implemented in the prototype it was analyzed and added to the broader concept of a component-based visualization system (see 5).

Requirement 100 and 120 were not specifically mentioned in this document. They are both not an integral part of a new concept for a data visualization system and are thus not part of this document. But they are valuable to the stakeholders. The other two noteworthy requirements are FR 040 and 110. They are partly implemented in DAVID. This means for FR 040 (...1) that the linking of the visual behavior is not completely implemented. When zooming into a line chart for example, the linked components do not reproduce this behavior. FR 110 (...2) is implemented by a database friendly export. The system exports annotations in the form of a CSV[1] but does not save automatically into a database.

8.2 FUTURE WORK

In 6.2.2 and 5.4.1, the concept of library based queries is introduced. This concept is valuable, as it allows users to rapidly begin the process of data visualization. However, it

1 Comma Separated Values

FR #	Name	Concept	DAVID
010	Separation of concerns	✓	✓
020	Animations	✓	✓
030	Highlighting	✓	✓
040	Focusing and Linking	✓	. . . 1
050	Linking components and data	✓	✓
060	Overlaying meta-information	✓	✓
070	Annotating visualized data	✓	✓
080	Freedom of choice	✓	✓
090	Creative freedom	✓	✓
100	Sharing of workspaces	✗	✗
110	Saving of results	✓	. . . 2
120	Video recording of workspace	✗	✗
130	User-friendly data queries	✓	✓

Table 4: Comparison of initial and fulfilled requirements

also limits the user's freedom, which I argue is a considerable limitation (see 5.2.2). The current implementation uses the "dataManipulator" class to alter data on the fly once it is received. These manipulators can be defined while configuring the data calls (see 7.1 and 31). But manipulators can only work on data that has already been received, and require domain knowledge of the scripting language. These weaknesses need to be addressed in the future. One solution may be to make heavier use of "flow based programming" and introduce a pipe and filter architecture for data fetching. The altering of data needs to be as straightforward and user friendly as the current visualization process is. I also propose to incorporate heavy computations and data selection strategies that are run before the visualization starts [72].

Another issue is DAVID's way of styling visualizations. DAVID relies heavily on third party libraries chosen by the user to visualize data. This also makes the user responsible for defining the aesthetics of a visualization. While many libraries provide ways of generating standard charts, such as a line chart, they fail to color or label them with an eye toward optimal legibility. This may cause problems of accessibility (for example red-green blindness), as well as when interpreting data generally. The authors of [50] argue that for standard visualizations, "semantically reasonable" colors can be chosen automatically. Such a mechanism would enrich DAVID's visualization process and enhance its user-friendliness. DAVID is not currently capable of suggesting best practices to the user.

Most importantly, however, the effectiveness and efficiency of the system has not undergone a larger test. Though it was used in production-like scenarios, the evaluation of this software is a project on its own, especially due to the rich feature set DAVID supplies. Even though I argue that DAVID allows data-analysts to rapidly experience and interact with data in an efficient manner, this claim is not yet proven for a diverse or representative set of use cases. Such an evaluation is beyond the scope of this work.

BIBLIOGRAPHY

[1] The mit agelab homepage. URL http://agelab.mit.edu/about-agelab. (Cited on page 9.)

[2] Data visualization and infographics, Jan. 2008. URL http://www.smashingmagazine.com/2008/01/14/monday-inspiration-data-visualization-and-infographics/.

[3] Dart language perfomance metrics, March 2014. URL https://www.dartlang.org/performance/. (Cited on pages xi and 59.)

[4] 5 charts that show apple's plateau after a meteoric rise, 2014. URL http://www.forbes.com/sites/markrogowsky/2013/10/11/5-charts-that-show-apples-plateau-after-a-meteoric-rise/. (Cited on pages xi and 29.)

[5] Data-driven documents, 2014. URL http://d3js.org//. (Cited on pages xi, 60, and 61.)

[6] Highcharts 3.0, 2014. URL http://www.highcharts.com/. (Cited on pages xi, 60, and 61.)

[7] Lawsuits in the mobile business, 2014. URL http://news.designlanguage.com/post/1473307539. (Cited on pages xi and 30.)

[8] Mental models, March 2014. URL http://www.nngroup.com/articles/mental-models/. (Cited on page 64.)

[9] Processingjs, 2014. URL http://processingjs.org/reference/. (Cited on pages xi and 61.)

[10] Angular js, 2014. URL https://angularjs.org/. (Cited on pages xi and 49.)

[11] Ducksboard - all of your data. in one place., 2014. URL https://ducksboard.com/. (Cited on pages xi and 8.)

[12] Raphaeljs, 2014. URL http://raphaeljs.com/. (Cited on pages xi and 61.)

[13] Rstudio project page, 2014. URL http://www.rstudio.com/products/rstudio/. (Cited on pages xi and 7.)

[14] Tableau - rapid-fire business intelligence. in the cloud., 2014. URL http://www.tableausoftware.com/products/online. (Cited on pages xi and 8.)

[15] ISO/IEC/IEEE 42010. A conceptual model of architecture description, 2011. URL http://www.iso-architecture.org/ieee-1471/cm/. [Online; accessed 21-March-2014].

[16] David A. Aoyama, Jen-Ting T. Hsiao, Alfonso F. Cárdenas, and Raymond K. Pon. Timeline and visualization of multiple-data sets and the visualization querying challenge. *J. Vis. Lang. Comput.*, 18(1):1–21, February 2007. ISSN 1045-926X. doi: 10.1016/j.jvlc.2005.11.002. URL http://dx.doi.org/10.1016/j.jvlc.2005.11.002.

[17] Henry C. Baker, Jr. and Carl Hewitt. The incremental garbage collection of processes. *SIGPLAN Not.*, 12(8):55–59, August 1977. ISSN 0362-1340. doi: 10.1145/872734.806932. URL http://doi.acm.org/10.1145/872734.806932. (Cited on page 41.)

[18] David Benyon. *Designing Interactive Systems*. Pearson, 2 edition, 2010. (Cited on page 66.)

[19] Bondi and B AndrÃ©. Characteristics of scalability and their impact on performance. In *Proceedings of the second international workshop on Software and performance - WOSP*, 2000. ISBN 158113195. (Cited on page 12.)

[20] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, December 2011. ISSN 1077-2626. doi: 10.1109/TVCG.2011.185. URL http://dx.doi.org/10.1109/TVCG.2011.185. (Cited on page 20.)

[21] Encyclopedia Britannica. System, 2014. URL http://www.merriam-webster.com/dictionary/system. (Cited on page 19.)

[22] Bernd Bruegge. *Objektorientierte Softwaretechnik mit UML, Entwurfsmuster und Java*. Prentice Hall, 2004.

[23] Bernd Bruegge. *Design Patterns lecture*. 3rd edition, 2012. (Cited on page 45.)

[24] Andreas Buja, John Alan McDonald, John Michalak, and Werner Stuetzle. Interactive data visualization using focusing and linking. In *Proceedings of the 2Nd Conference on Visualization '91*, VIS '91, pages 156–163, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press. ISBN 0-8186-2245-8. URL http://dl.acm.org/citation.cfm?id=949607.949633. (Cited on pages 20 and 27.)

[25] Hong Chen. Towards design patterns for dynamic analytical data visualization. In *Proceedings Of SPIE Visualization and Data Analysis*, pages 75–86, 2004. (Cited on page 20.)

[26] M. Chen, D. Ebert, H. Hagen, R.S. Laramee, R. Van Liere, K.-L. Ma, W. Ribarsky, G. Scheuermann, and D. Silver. Data, information, and knowledge in visualization. *Computer Graphics and Applications, IEEE*, 29(1):12–19, Jan 2009. ISSN 0272-1716. doi: 10.1109/MCG.2009.6. (Cited on pages xi, xii, 3, 6, 21, and 25.)

[27] Ed H. Chi. A taxonomy of visualization techniques using the data state reference model. *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, 2000. (Cited on pages 21 and 34.)

[28] Hank Childs, Eric Brugger, Kathleen Bonnell, Jeremy Meredith, Mark Miller, Braid Whitlock, and Nelson Max. A contract based system for lage data visualization. *IEEE Visualization*, October 2005. (Cited on page 53.)

[29] William S. Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):pp. 531–554, 1984. ISSN 01621459. URL http://www.jstor.org/stable/2288400. (Cited on page 35.)

[30] William T. Councill and George T. Heineman. *Component-Based Software Engineering*. Addison-Wesley, 2001.

[31] Alan M. Davis. Operational prototyping: A new development approach. *IEEE Software*, page 71, September 1992. (Cited on page 31.)

[32] Dmitry Fadeyev. 10 useful techniques to improve your user interface design, December 2008. URL http://uxdesign.smashingmagazine.com/2008/12/15/10-useful-techniques-to-improve-your-user-interface-designs/.

[33] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[34] Freeman. *Entwurfsmuster von Kopf bis Fuß*. O'Reilly, 4 edition, 2008.

[35] Michael Friendly. Milestones in the history of thematic cartography, statistical graphics, and data visualization, 2008. (Cited on page 3.)

[36] Erich Gamme, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns : Elements of Reusable Object Oriented Software. 1995. (Cited on page 55.)

[37] Hector Gonzalez, Alon Halevy, Christian S. Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, and Warren Shen. Google fusion tables: Data management, integration and collaboration in the cloud. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 175–180, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0036-0. doi: 10.1145/1807128.1807158. URL http://doi.acm.org/10.1145/1807128.1807158.

[38] Jeffrey Heer and Maneesh Agrawala. Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):853–860, September 2006. ISSN 1077-2626. doi: 10.1109/TVCG.2006.178. URL http://dx.doi.org/10.1109/TVCG.2006.178. (Cited on pages xi, 20, 41, and 51.)

[39] Jeffrey Heer and Michael Bostock. Declarative language design for interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1149–1156, November 2010. ISSN 1077-2626. doi: 10.1109/TVCG.2010.144. URL http://dx.doi.org/10.1109/TVCG.2010.144.

[40] Jeffrey Heer, Maneesh Agrawala, and Wesley Willett. Generalized selection via interactive query relaxation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 959–968, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1. doi: 10.1145/1357054.1357203. URL http://doi.acm.org/10.1145/1357054.1357203.

[41] Jeffrey Heer, Nicholas Kong, and Maneesh Agrawala. Sizing the horizon: The effects of chart size and layering on the graphical perception of time series visualizations.

In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1303–1312, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-246-7. doi: 10.1145/1518701.1518897. URL `http://doi.acm.org/10.1145/1518701.1518897`.

[42] Jefrey Heer. Animated transitions in statistical data graphics jeffrey heer. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1240–1247, nov November 2007. (Cited on pages 20, 27, and 38.)

[43] *A Distributed Blackboard Architecture for Interactive Data Visualization*, 1998. IEEE, IEEE Visualization.

[44] Noah Iliinsky and Julie Steele. *Designing Data Visualizations*. OReilly, 2011. (Cited on pages xi, 25, 26, 27, and 28.)

[45] T.J.; Jankun-Kelly and Kwan-Liu Ma. A spreadsheet interface for visualization exploration.

[46] Daniel A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, January 2002. ISSN 1077-2626. doi: 10.1109/2945.981847. URL `http://dx.doi.org/10.1109/2945.981847`. (Cited on page 3.)

[47] Nicholas Kong and Maneesh Agrawala. Graphical overlays: Using layered elements to aid chart reading. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2631–2638, 2012. URL `http://dblp.uni-trier.de/db/journals/tvcg/tvcg18.html#KongA12`. (Cited on pages xi, 14, 20, 35, and 36.)

[48] H Krcmar. Informationsmanagement, 2012. Lecture at TUM, Germany. (Cited on page 3.)

[49] KentD. Lee. Event-driven programming. In *Python Programming Fundamentals*, Undergraduate Topics in Computer Science, pages 149–165. Springer London, 2011. ISBN 978-1-84996-536-1. doi: 10.1007/978-1-84996-537-8_6. URL `http://dx.doi.org/10.1007/978-1-84996-537-8_6`. (Cited on page 37.)

[50] Sharon Lin, Julie Fortuna, Chinmay Kulkarni, Maureen Stone, and Jeffrey Heer. Selecting semantically-resonant colors for data visualization. *Computer Graphics Forum (Proc. EuroVis)*, 2013. URL `http://vis.stanford.edu/papers/semantically-resonant-colors`. (Cited on page 76.)

[51] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. immens: Real-time visual querying of big data. *Computer Graphics Forum (Proc. EuroVis)*, 32, 2013. URL `http://vis.stanford.edu/papers/immens`. (Cited on pages 19 and 53.)

[52] Jayant Madhavan, Sreeram Balakrishnan, Kathryn Brisbin, Hector Gonzalez, Nitin Gupta, Alon Halevy, Karen Jacqmin-Adams, Heidi Lam, Anno Langen, Hongrae Lee, Rod McChesney, Rebecca Shapley, and Warren Shen. Big data storytelling through interactive maps. *IEEE Computer Society Technical Committee on Data Engineering*, 2012. (Cited on page 20.)

[53] Robert C. Martin. Design principles and design patterns. 2000. (Cited on pages 49 and 50.)

[54] MATLAB. Matlab - the language of technical computing, 2014. URL http://www.mathworks.com/products/matlab/index.html. [Online; accessed 31-March-2014]. (Cited on pages xi and 7.)

[55] Florian Matthes. Software architectures. Lecture, 2011. (Cited on pages xi, 9, 33, 47, and 48.)

[56] Matt McKeon. Harnessing the information ecosystem with wiki-based visualization dashboards. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1081–1088, November 2009. ISSN 1077-2626. doi: 10.1109/TVCG.2009.148. URL http://dx.doi.org/10.1109/TVCG.2009.148.

[57] Bertrand Meyer. *Object-Oriented Software Construction*. ISE, 2 edition, 2000. (Cited on pages xii, 32, and 37.)

[58] Miriah Meyer, Michael Sedlmair, and Tamara Munzner. The four-level nested model revisited: Blocks and guidelines. In *Proceedings of the 2012 BELIV Workshop: Beyond Time and Errors - Novel Evaluation Methods for Visualization*, BELIV '12, pages 11:1–11:6, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1791-7. doi: 10.1145/2442576.2442587. URL http://doi.acm.org/10.1145/2442576.2442587.

[59] M. Michael, J.E. Moreira, D. Shiloach, and R.W. Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, March 2007. doi: 10.1109/IPDPS.2007.370631. (Cited on page 13.)

[60] Hirotaka Mizuno, Yuichi Mori, Yoji Taniguchi, and Hiroshi Tsuji. Data queries using data visualization techniques. *IEEE*, 1997.

[61] Scott Murray. *Interactive Data Visualization for the Web*. OReilly, 2 edition, 2013. (Cited on pages 25 and 26.)

[62] Chris North and Ben Shneiderman. Snap-together visualization: A user interface for coordinating visualizations via relational schemata. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, pages 128–135, New York, NY, USA, 2000. ACM. ISBN 1-58113-252-2. doi: 10.1145/345513.345282. URL http://doi.acm.org/10.1145/345513.345282. (Cited on pages 19 and 41.)

[63] Object-Managment-Group. *UML Superstructure Specification, v2.1.2*. Object Management Group, Framingham, Massachusetts, November 2007.

[64] Martin Odersky, Erik Meijer, and Roland Kuhn. Principles of reactive programming, 2014. URL https://www.coursera.org/course/reactive. Online course of the EPA.

[65] P. Rheingans. Are we there yet? exploring with dynamic visualization. *IEEE Comput. Graph. Appl.*, 22(1):6–10, January 2002. ISSN 0272-1716. doi: 10.1109/38.974511. URL http://dx.doi.org/10.1109/38.974511.

[66] Jennifer Rowley. *The wisdom hierarchy: representations of the DIKW hierarchy*. 2007. (Cited on page 6.)

[67] Robert Scoble and Shel Israel. *Age of Context*. Patrick Brewster Press, 2014. (Cited on page 3.)

[68] Perceptual Edge Stephen Few. Time on the horizon, June/July 2008. Visual Business Intelligence Newsletter. (Cited on page 19.)

[69] Lloyd A. Treinish. Task-specific visualization design: A case study in operational weather forecasting. In *Proceedings of the Conference on Visualization '98*, VIS '98, pages 405–409, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press. ISBN 1-58113-106-2. URL http://dl.acm.org/citation.cfm?id=288216.288330.

[70] Birgit Vogel-Heuser. Industrielle software-entwicklung fuer ingenieure, 2013. Lecture at TUM, Germany.

[71] W3C. Shadow dom, 2014. URL http://w3c.github.io/webcomponents/spec/shadow/. (Cited on page 56.)

[72] Hadley Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1):1–29, 4 2011. ISSN 1548-7660. URL http://www.jstatsoft.org/v40/i01. (Cited on page 76.)

[73] Wikipedia. Extensibility — wikipedia, the free encyclopedia, 2013. URL http://en.wikipedia.org/w/index.php?title=Extensibility&oldid=573614867. [Online; accessed 21-March-2014]. (Cited on page 12.)

[74] Wikipedia. Ubiquitous computing — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Ubiquitous_computing&oldid=613691982. [Online; accessed 27-June-2014]. (Cited on page 3.)

[75] Wikipedia. Big data — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Big_data&oldid=600337476. [Online; accessed 19-March-2014]. (Cited on page 13.)

[76] Wikipedia. Component-based software engineering — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Component-based_software_engineering&oldid=600742295. [Online; accessed 22-March-2014]. (Cited on page 32.)

[77] Wikipedia. Cascading style sheets — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Cascading_Style_Sheets&oldid=603295455. [Online; accessed 9-April-2014]. (Cited on page 56.)

[78] Wikipedia. Data visualization — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Data_visualization&oldid=596329588. [Online; accessed 19-March-2014]. (Cited on page 7.)

[79] Wikipedia. Digital revolution — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Digital_Revolution&oldid=598028059. [Online; accessed 21-March-2014]. (Cited on page 3.)

[80] Wikipedia. Flow-based programming — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Flow-based_programming&oldid=594528260. [Online; accessed 23-March-2014]. (Cited on page 37.)

[81] Wikipedia. Futures and promises — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Futures_and_promises&oldid=600538588. [Online; accessed 23-March-2014]. (Cited on page 41.)

[82] Wikipedia. Information age — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Information_Age&oldid=600520556. [Online; accessed 21-March-2014]. (Cited on page 3.)

[83] Wikipedia. Marshalling (computer science) — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Marshalling_(computer_science)&oldid=601087005. [Online; accessed 31-March-2014]. (Cited on page 52.)

[84] Wikipedia. Modular programming — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Modular_programming&oldid=600009554. [Online; accessed 22-March-2014]. (Cited on page 32.)

[85] Wikipedia. Computer performance — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Computer_performance&oldid=590334974. [Online; accessed 21-March-2014]. (Cited on page 12.)

[86] Wikipedia. Software portability — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Software_portability&oldid=599748181. [Online; accessed 21-March-2014]. (Cited on page 12.)

[87] Wikipedia. Software prototyping — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Software_prototyping&oldid=595274215. [Online; accessed 22-March-2014]. (Cited on page 31.)

[88] Wikipedia. Scientific method — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Scientific_method&oldid=599211918. [Online; accessed 19-March-2014].

[89] Wikipedia. Separation of concerns — wikipedia, the free encyclopedia, 2014. URL http://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=599843173. [Online; accessed 30-March-2014]. (Cited on pages 14 and 50.)

[90] Wesley Willett, Jeffrey Heer, Joseph Hellerstein, and Maneesh Agrawala. Commentspace: Structured support for collaborative visual analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 3131–3140, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0228-9. doi: 10.1145/1978942.1979407. URL http://doi.acm.org/10.1145/1978942.1979407.